H. Tornsdorf
M. Tornsdorf

64

C64 BASIC für Einsteiger

DATA BECKER

Helmut Tornsdorf
Manfred Tornsdorf

C64 BASIC für Einsteiger

DATA BECKER

2. unveränderte Auflage 1989

ISBN 3-89011-246-3

Copyright © 1988

DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Umschlaggestaltung: Werner Leinhos

Text verarbeitet mit Word 4.0, Microsoft
Ausgedruckt mit Hewlett Packard LaserJet II
Druck und Verarbeitung: Graf & Plügge, Düsseldorf

Alle Rechte vorbehalten

Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle technischen Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

Wichtiger Hinweis:

Die in diesem Buch wiedergegebenen Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle technischen Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

Auf ein Vor-Wort

Was dieses Buch Ihnen bietet

Herzlichen Glückwunsch. Sie haben sich mit dem C64 den Erfolgscomputer der 80er Jahre gekauft. Sie haben damit nicht nur Zugang zu einer kaum übersehbaren Menge an Programmen, sondern Sie haben auch einen Computer, der wie wenige andere den Benutzer geradezu herausfordert, sich intensiv mit ihm zu beschäftigen. Der Schlüssel dazu ist beim C64 schon fest eingebaut und heißt BASIC.

Dieses BASIC möchten wir Ihnen in diesem Buch vorstellen. Keine Angst: Sie werden dabei eine Menge lernen, aber dieses Lernen wird mehr so nebenbei passieren. Im Vordergrund soll nämlich der Spaß an der Beschäftigung mit dem C64 und seinem BASIC stehen.

Damit der Spaß garantiert ist und der Lerneffekt trotzdem so groß wie möglich ist, haben wir uns eine Menge einfallen lassen:

1. Wir wollen uns nicht lange mit allgemeinen Erörterungen und Erklärungen aufhalten, sondern Sie schnell zu ersten Erfolgserlebnissen führen.
2. Damit Sie keine Schwierigkeiten haben und nie den Spaß verlieren, wollen wir Schritt für Schritt vorgehen.
3. Durch Wiederholungen und Zusammenfassungen werden die neuen Kenntnisse und Erfahrungen schnell zum sicheren Besitz.
4. Bei den (vielen) Beispielen handelt es sich um Lösungen für Alltagsprobleme, die Sie leicht für Ihre persönlichen Interessen abwandeln können.
5. Der größte Spaß am Computer kann einem schnell vergehen, wenn man in einer Situation einfach nicht mehr weiter weiß. Davor kann auch eine sorgfältige Vorgehensweise und Erklärung nicht schützen. Damit Sie aber unter sol-

chen Pannen nicht leiden müssen, haben wir ein spezielles Kapitel in diesem Buch eingerichtet: den Pannenservice. Wenn bei der Arbeit mit diesem Buch irgend etwas nicht so funktioniert wie beschrieben, sollten Sie auf jeden Fall in diesem Kapitel Rat suchen.

6. Sie werden mit Hilfe dieses Buches schon bald kein Einsteiger mehr sein. Damit Sie möglichst viel für Ihr Geld bekommen, haben wir noch zwei Teile eingebaut, die Ihnen auch später noch von großem Nutzen sein werden. In einem Aufsteigerteil zeigen wir Ihnen verschiedene Möglichkeiten, wie es nach diesem Buch am besten weitergehen kann. Noch länger werden Sie wahrscheinlich das Lexikon am Ende dieses Buches nutzen. Es enthält in knapper Form noch einmal alle wichtigen Informationen dieses Buches, nur eben alphabetisch und damit leicht greifbar.

Um Ihnen einen kleinen Einblick von den Themen dieses Buches zu geben, wollen wir einige noch besonders hervorheben:

Sequentielle Dateien

Wir haben uns in diesem Einsteigerbuch nicht damit begnügt, Ihnen nur die BASIC-Grundbefehle vorzustellen. Vielmehr haben wir zusätzlich die Themen aufgenommen, die erfahrungsgemäß Einsteigern (übrigens damals auch uns) große Schwierigkeiten gemacht haben. Dazu gehört besonders der Bereich der "sequentiellen Dateien", das heißt: Speicherung einzelner Daten auf Diskette. Wir versprechen Ihnen, daß Sie nur einen Bruchteil der Zeit brauchen werden, die wir damals benötigt haben.

ASCII-Grafik

In vielen Büchern wird nicht auf das Thema Grafik eingegangen, oder es wird gleich sehr schwierig. Wir zeigen Ihnen, wie man ganz einfach Grafiken erstellen kann, und nutzen dazu die normale Tastatur des C64.

HIRES-Grafik

Natürlich gehen wir in einem späteren Kapitel auch auf eine besondere Fähigkeit des C64 ein, die sogenannte hochauflösende Grafik. Mit ihr können Sie punktgenaue, detailreiche Bilder er-

stellen (z.B. mathematische Funktionen darstellen). Vielleicht haben Sie auch schon einmal gehört, daß die konkrete Benutzung dieser Grafik nicht ganz problemlos ist. Wir haben uns einfache Lösungen einfallen lassen.

Sprites

Gerade der C64 ist neben vielen nützlichen Anwendungen auch für Spiel und Spaß geeignet. In diesem Zusammenhang kommt man an den sogenannten Sprites gar nicht vorbei. Am Ende des entsprechenden Kapitels wissen Sie nicht nur, wie Ballons und ähnliche kleine Grafik-Objekte über den Bildschirm bewegt werden. Sie haben auch nützliche Programme, die den Einbau von Sprites in Ihre eigenen Programme mächtig erleichtern. Den versprochenen Spaß bei der Benutzung dieses Buches werden Sie sicherlich haben, wenn Sie mehrmals erfolgreich versucht haben, auch bei widrigen Windverhältnissen einen Ballon sicher zu landen (siehe dazu das Programm in Kapitel 9).

Wir hoffen, daß Sie am Ende dieses Buches mit uns der Meinung sind, daß der Einstieg in die Welt des Programmierens mit BASIC gar nicht kompliziert und schwierig sein muß. Uns bleibt nur noch, Ihnen zusätzlich viel Freude und Spaß bei der Arbeit mit diesem Buch und dem C64 zu wünschen.

Münster, im Mai 1988 *Die Autoren*

Inhaltsverzeichnis

Auf ein Vor-Wort 7

1. Allgemeines und Vorarbeiten 15

1.1 C64 - das heißt BASIC 15
1.2 Was man vom C64 und seiner Tastatur wissen sollte 17
1.3 Die Vorbereitung von Disketten 20

2. BASIC an einem Abend: Schnell zum ersten Programm 23

2.1 Der erste Kontakt mit BASIC 23
2.2 BASIC bekommt ein Gedächtnis: Das erste Programm 32
2.3 Programme auf Diskette: Speichern, Suchen, Laden, Löschen 37
2.4 Der Computer stellt Fragen: INPUT 42
2.5 Programme schwarz auf weiß: Das Ausdrucken 45
2.6 Zusammenfassung 47
2.7 BASIC als erste Hilfe für den Alltag 49

3. BASIC an einem Wochenende 53

3.1 Schubladen und Platzhalter: Variablen für Zahlen in BASIC 53
3.2 Variablen können auch Wörter enthalten: Stringvariablen 56
3.3 Wie man sich lästige Tipperei erspart: Das Kopieren und Löschen von Programmzeilen 60
3.4 Zusammenfassung 61

3.5 Übungen zur Sicherung des Erreichten 63
3.5.1 Mehrwertsteuer 63
3.5.2 Rabatt 64
3.5.3 Die Gehaltserhöhung 66

4. Was man für größere Programme noch braucht: Weitere wichtige BASIC-Befehle 69

4.1 BASIC kann springen: GOTO 69
4.2 BASIC erfüllt Bedingungen: IF ... THEN 70
4.3 BASIC läßt sich Grenzen setzen: FOR ... NEXT 74
4.4 Zusammenfassung 75

5. Es geht nichts über die Praxis! Anwendung der neuen Befehle 77

5.1 Erste Anwendung der neuen Befehle: Die Benzinkostentabelle 77
5.2 Zweite Anwendung der neuen Befehle: Zahlenraten 83
5.3 Dritte Anwendung der neuen Befehle: Das große Einmaleins 88
5.4 Falls mal kein Würfel da ist: Kein Problem mit BASIC 90
5.5 Zusammenfassung 90

6. Einfache Grafik und andere nützliche Dinge 93

6.1 Grafik ganz einfach: Die ASCII-Grafik 93
6.2 Sternenhimmel 96
6.3 Unterprogramme: GOSUB und RETURN 99
6.4 Malen mit dem Computer 101
6.5 Balken 107
6.6 Der Anfang für das Programm Balkengrafik 114
6.7 Das fertige Programm BALKENGRAFIK 120
6.8 Zusammenfassung 123

7. Daten in Programmen **127**

7.1 So einfach kommt ein Programm an seine Daten: READ und DATA 127
7.2 Felder - viele Daten unter einem Namen 131
7.3 BALKENGRAFIK mit Feldern 134
7.4 Zwischen-Zusammenfassung 137
7.5 Daten auf Diskette - sequentielle Dateien 139
7.6 Was bei der Datenspeicherung auf Diskette zu beachten ist 139
7.7 So kommen die Daten für die Balken auf Diskette .. 144
7.8 Das Programm BALKENGRAFIK mit Diskettenspeicherung 148
7.9 Zusammenfassung 149

8. Die hochauflösende Grafik **153**

8.1 Die Besonderheiten der hochauflösenden Grafik 153
8.2 Ein- und Ausschalten der hochauflösenden Grafik .. 155
8.3 Sauberer Bildschirm bei hochauflösender Grafik 157
8.4 Punkte setzen 159
8.5 Ein Mini-Malprogramm 162
8.6 Mathematik in höchster Auflösung 164
8.7 Zusammenfassung 167

9. Sprites **171**

9.1 Was sind Sprites? 171
9.2 Das erste Sprite 172
9.3 Wir erstellen ein Sprite 177
9.4 Der Ballon 179
9.5 Sprites bewegen 181
9.6 Das Ballonspiel 184
9.7 Zusammenfassung 191

10. Ausblicke für Aufsteiger 193

10.1 Weiterführende Literatur 193
10.2 Weitere Möglichkeiten von BASIC 195
10.3 So wird BASIC schneller - Compiler 197

11. Wenn etwas schiefgeht: Unser Pannen-Service 199

Lexikon zum BASIC des C64 209

Stichwortverzeichnis 241

1. Allgemeines und Vorarbeiten

Natürlich möchten Sie jetzt sofort mit dem Einstieg in BASIC anfangen, aber vorher wollen wir noch einige unklare Begriffe klären und notwendige Vorarbeiten erledigen. Es ist nämlich sehr wichtig, daß wir für unser gemeinsames Vorhaben in etwa den gleichen Stand haben. Nur dann können Sie allen unseren Schritten folgen, und Frust und Streß bleiben von vornherein für Sie Fremdwörter.

1.1 C64 - das heißt BASIC

Wenn Sie Ihren C64 nicht gerade erst gekauft haben, dann haben Sie den ersten Kontakt mit BASIC eigentlich schon hinter sich. Nach dem Einschalten meldet sich der Computer nämlich mit:

```
    **** COMMODORE 64 BASIC V2 ****

 64K RAM SYSTEM 38911 BASIC BYTES FREE

 READY.
```

Dies ist gleichermaßen das erste Lebenszeichen Ihres C64 und auch des mitgelieferten BASICs. Im Gegensatz zu vielen anderen Computern (Atari ST, Amiga, PC) muß dieses BASIC nämlich nicht etwa von der Diskette geladen werden, sondern befindet sich fest eingebaut im Computer. Es steht damit sofort nach dem Einschalten zur Verfügung. Wenn Sie nun irgendeine sinnvolle Eingabe machen, mit der Ihr C64 etwas anfangen kann, dann ist das auch gleichzeitig ein BASIC-Befehl. Allein schon deswegen nimmt BASIC eine ganz wichtige Stellung beim C64 ein. Hier gibt es nicht wie bei anderen Computern erst einmal ein Betriebssystem, mit dem man direkt arbeiten kann, um sich erst später mit einer Programmiersprache zu beschäftigen.

Allerdings können wir uns recht lange sinnvoll mit dem C64 beschäftigen, ohne im eigentlichen Sinne zu programmieren. Schon ein einfacher Ladebefehl genügt, um Texte zu schreiben, Bilder zu erstellen, Daten zu verwalten oder die Zeit mit einem

spannenden Spiel zu verbringen - immer vorausgesetzt, man hat das zugehörige Programm erworben.

GEOS

Eine besondere Stellung für den C64 nimmt unserer Meinung nach GEOS ein. Mit GEOS haben wir wirklich ein praktisch völlig selbständiges Betriebssystem, mit dem wir fast alle anfallenden Aufgaben lösen können, ohne mit dem eingebauten BASIC in Kontakt zu kommen. Dabei ist GEOS recht einfach zu bedienen und bietet viele Möglichkeiten, die man sonst erst bei wesentlich teureren Computern findet.

Viele Aufgaben lassen sich aber immer noch am einfachsten direkt mit dem C64 und seinem BASIC lösen. Mit wenigen Programmzeilen sind beispielsweise einige Unkosten zusammengerechnet, aber auch eine SINUS-Kurve auf dem Bildschirm erstellt. In so einem Fall lohnt es kaum, erst ein Programm zu laden, seine Bedienung kennenzulernen und dann das Problem zu lösen. Häufig findet man auch für spezielle Aufgaben, die man lösen möchte, gar nicht das geeignete Programm. Und dann ist es schon sehr gut, wenn man selbst programmieren kann.

Zum anderen ist es unserer Meinung nach wichtig, sich nicht ausschließlich auf der "Oberfläche" eines Computers zu bewegen. Wenn wir nur wissen, welche Tasten wir drücken und welche Diskette wir benutzen müssen, dann wird es immer wieder Situationen geben, in denen wir einfach nicht weiterwissen. Wissen wir aber in etwa, wie ein Computer funktioniert und was in groben Zügen in ihm abläuft, dann können wir uns in fast allen Situationen mit diesem Wissen weiterhelfen. Dies können wir dadurch erreichen, daß wir "seine" Sprache lernen, uns mit ihm verständigen können. Programmieren ist ein wichtiger Schritt auf dem Wege zum Kennenlernen - und damit zum Beherrschen eines Computers.

Natürlich wollen wir dabei zusätzlich noch eine Menge Spaß haben. Es gibt schon genug Situationen, in denen man etwas lernen muß, und vieles hat man dann nach kurzer Zeit wieder vergessen. Wir wollen Ihnen in den nächsten Kapiteln möglichst viel

von unserem eigenen Spaß am Programmieren weitergeben. Und Sie wissen doch, das schönste Lernen ist das, bei dem man es gar nicht merkt, sondern hinterher nur staunend sieht, daß man es kann.

1.2 Was man vom C64 und seiner Tastatur wissen sollte

Bevor wir nun in das Abenteuer Programmieren einsteigen, wollen wir noch zwei ganz kurze Vorbereitungen treffen, damit später erst gar keine Probleme auftauchen können. Wir wollen ein paar wichtige Begriffe klären, die uns in den ersten Kapiteln ständig begegnen werden, und auf ein paar Besonderheiten hinweisen. Sollten Sie sich schon etwas sicher im Umgang mit dem C64 fühlen, so können Sie die Kapitel 1.2 und 1.3 überschlagen, sollten aber anschließend eine leere, formatierte Diskette zur Verfügung haben.

Wenn wir mit einer neuen Sache anfangen, dann gehen wir immer davon aus, daß sich Ihr C64 im Grundzustand befindet, als wäre er gerade eben erst eingeschaltet worden. Sollten Sie vorher irgend etwas mit ihm gemacht haben und sich nicht sicher sein, ob er wirklich im Grundzustand ist, dann schalten Sie ihn bitte kurz aus und wieder ein. Es wäre doch schade, wenn bei Ihnen etwas nicht wie beschrieben funktioniert, nur weil Sie vorher ein Programm geladen haben, das irgend etwas Wichtiges geändert hat.

Wir möchten Sie bitten, möglichst alle Schritte genauso wie im Buch beschrieben nachzumachen, es sei denn, wir weisen ausdrücklich darauf hin, daß etwas übersprungen werden kann. Wir haben uns nämlich bei den Schritten und ihrer Reihenfolge eine Menge gedacht, und oft funktioniert ein Schritt nur dann, wenn der vorherige auch erfolgt ist. Später in diesem Buch werden Sie Ihren C64 besser kennen und selbst entscheiden können, was unbedingt notwendig ist und was nicht.

Die Return-Taste - und Ihr C64 weiß, wenn's losgeht

Wir werden in diesem Buch viele Eingaben machen, indem wir etwas per Tastatur auf den Bildschirm schreiben. Meist muß anschließend eine spezielle Taste gedrückt werden, damit der Computer weiß, daß wir fertig sind. Diese Taste befindet sich ziemlich weit rechts auf der Tastatur und trägt die Aufschrift "Return". Zu Anfang werden wir häufig hinter die eigentliche Eingabe einen entsprechenden Hinweis schreiben, also etwa:

```
PRINT "HELLO" (plus Return-Taste)
```

oder auch

```
PRINT "HELLO" (Return)
```

Bitte geben Sie diese Hinweise nicht wie die davor stehenden Befehle ein, sondern drücken Sie dann einfach die entsprechende Taste.

Umschaltung der Tastatur und wie sie wieder "normal" wird

Wenn Sie Ihren C64 gerade eingeschaltet haben und einige Buchstaben drücken, so werden diese als Großbuchstaben auf den Bildschirm geschrieben. Bei einer Schreibmaschine würden dagegen kleine Buchstaben erscheinen, und Sie müßten erst zusätzlich den Wagen anheben, um große Buchstaben zu erzielen. Zu diesem Zweck gibt es beim C64 die beiden Shift-Tasten, die sich links und rechts unten neben der Leer-Taste (auch Space-Taste genannt) befinden. Wenn Sie dagegen eine Shift-Taste festhalten und einen Buchstaben drücken, erhalten Sie ganz eigenartige Zeichen. Wir wollen uns am Anfang darum nicht weiter kümmern, aber auf eine Besonderheit hinweisen, die Ihnen am Anfang große Schwierigkeiten machen könnte:

Drücken Sie bitte einmal (in etwa gleichzeitig) zwei Tasten links unten auf der Tastatur, und zwar Shift und die Taste links davon, die Commodore-Taste. Wir wollen sie ab jetzt entsprechend ihrer Aufschrift C=-Taste nennen. Wenn noch Zeichen auf Ihrem Bildschirm sind, dann verwandeln sich die Großbuchstaben in Kleinbuchstaben. Anschließend können Sie wie auf einer

Schreibmaschine mit großen und kleinen Buchstaben schreiben. Das erleichtert mit Sicherheit das Lesen. Wenn Sie einige Groß- und Kleinbuchstaben auf den Bildschirm geschrieben haben und nun wieder mit Hilfe der Shift- und C=-Taste umschalten, werden die kleinen Buchstaben zu großen. Aber: Die Neuerwerbung des C64, die Großbuchstaben neben den Kleinbuchstaben, wird zum Problem. Denn alle Großbuchstaben, die eben mit Hilfe der Shift-Taste erzeugt 'worden sind, werden jetzt zu seltsamen Bildchen, zu Grafikzeichen. Mit denen können wir zunächst nichts anfangen. Und von daher gehen wir auf Nummer sicher, d.h. in den Modus, in dem der C64 beim einfachen Schreiben (ohne Shift) immer Großbuchstaben anzeigt. Wenn wir jetzt einmal aus Versehen umschalten, haben wir immer noch eindeutig lesbare Anzeigen, wenn auch in kleinen Buchstaben.

Wenn in unserem Buch Eingaben großgeschrieben sind, so geben Sie diese also bitte ohne Shift ein - trotzdem erscheinen sie auf dem Bildschirm als Großbuchstaben. Sollten Sie plötzlich kleine Buchstaben auf dem Bildschirm erhalten, so haben Sie versehentlich die beschriebene Tastenkombination gedrückt und machen es bitte durch erneuten Druck rückgängig.

Wenn der C64 nicht mehr schreiben will

Es kann Ihnen passieren, daß der C64 sich scheinbar im Streik befindet und nicht mehr so reagiert wie erwartet. Sollten Sie beispielsweise den Cursor nicht mehr bewegen können und statt dessen eigenartige Zeichen erhalten, so befindet sich der C64 im sogenannten Einfüge- oder Gänsefüßchenmodus (Anführungsstriche). Am sichersten kommen Sie aus dem Dilemma heraus, indem Sie die Shift-Taste gedrückt halten und die Return-Taste drücken. Der C64 nimmt die Eingabe wegen der Verbindung mit der Shift-Taste nicht als Befehl und befindet sich in der nächsten Zeile wieder im gewohnten Modus.

Wenn es nicht mehr weitergeht: Unser Pannen-Service

Wir werden Ihnen dieses Problem weiter unten bei unseren praktischen Übungen noch genauer demonstrieren. An dieser Stelle sind wir schon darauf eingegangen, weil es zur Tastatur gehört und wir bei dieser Gelegenheit schon auf einen wichtigen Teil dieses Buches verweisen können:

Denn natürlich gibt es noch weitere Schwierigkeiten und Probleme, die immer mal wieder auftreten können. Diese haben wir für Sie an einer Stelle in diesem Buch mit Lösungsvorschlägen zusammengestellt. Schauen Sie doch bitte einmal ins Kapitel "Pannenhilfe".

1.3 Die Vorbereitung von Disketten

Computer sind von Natur aus sehr vergeßlich. Alles, was ihnen von uns mühsam beigebracht wurde, ist nach dem Ausschalten für immer verloren. Deshalb gibt es verschiedene Speichermöglichkeiten, die Daten dauerhaft festhalten: das Diskettenlaufwerk (Floppy) und den Kassettenrecorder (Datasette). Sollten Sie eine Datasette zum Speichern haben, dann besorgen Sie sich bitte mindestens zwei leere Kassetten. Als Besitzer eines Diskettenlaufwerks sollten Sie dementsprechend mindestens zwei leere Disketten haben. Haben Sie keine Speichermöglichkeit, dann lesen Sie bitte das Buch an einem Abend oder lassen Ihren C64 eingeschaltet oder - und das ist wohl die beste Lösung - Sie legen sich möglichst schnell eins der beiden Geräte zu, denn zum Programmieren gehört eine Möglichkeit des Speicherns wie das Benzin zum Autofahren.

Als Besitzer eines Diskettenlaufwerks - wir werden jetzt langsam auf den gebräuchlichen Begriff "Floppy" übergehen - müssen Sie die Diskette noch vorbereiten. Nehmen Sie bitte die schwarze Scheibe aus der Papierhülle, schieben Sie sie mit der glatten (meist beschrifteten) Seite nach oben in den Schlitz der Floppy, und schließen Sie diesen.

Achtung!

Der folgende Schritt ist vor der ersten Benutzung einer Diskette unerläßlich, führt aber unweigerlich zum Verlust jeder Information auf der Diskette. Dies ist aber bei einer neugekauften, leeren Diskette kein Problem, da sie noch keine Daten enthält.

An dieser Stelle können wie Ihnen leider nichts anderes anbieten, als eine ziemlich komplizierte Zeile abzutippen und damit das gewünschte Ergebnis zu erzielen. Geben Sie also bitte die nächste Zeile sorgfältig ein:

```
OPEN 1,8,15,"N:ARBEITSDISKETTE1,01":CLOSE1
```

Sollten Sie sich vertippen, so löschen Sie bitte mit der DEL-Taste (oben rechts auf der Tastatur) solange das letzte Zeichen, bis Sie fehlerfrei den Rest der Zeile schreiben können. Nach dem Drücken der Return-Taste haben Sie vielleicht zum ersten Mal das Gefühl, daß Ihr C64 lebt. Aus der Floppy kommt einige Sekunden lang ziemlicher Lärm, und anschließend brennt noch etwa 1.5 Minuten die rote Lampe der Floppy. Keine Sorge, Sie haben nichts falsch gemacht. So wie die Autowerkstatt zum Hammer greift, um den Kotflügel auszubeulen, greift die Floppy zu ziemlich rabiaten Methoden, um die Diskette für die spätere Arbeit vorzubereiten. Ein Grund zur Sorge besteht aber nicht.

Wenn die Lampe des Diskettenlaufwerks aus ist, entnehmen Sie die Diskette bitte und stecken sie gleich in die Papierhülle. Beschriften Sie einen der den Packungen beiliegenden Aufkleber z.B. mit "BASIC: ARBEITSDISKETTE1", und kleben Sie ihn auf die Diskette. Diese Diskette werden wir später dazu verwenden, unsere Programme zu speichern - legen Sie sie also an einem sicheren Ort griffbereit ab. Ein solcher sicherer Ort kann beispielsweise einer dieser Diskettenkästen sein, die es für wenig Geld gibt und die bei der bald wachsenden Zahl Ihrer Disketten schön Ordnung halten. Denn wenn Ihr Hund erst eine Diskette auf Schmackhaftigkeit geprüft oder ein Mitglied Ihrer Familie mehrere zum Ausgleich eines wackligen Tischbeins verwendet hat, ist es zu spät.

Nun aber nichts wie ran an BASIC und ans Programmieren!

2. BASIC an einem Abend: Schnell zum ersten Programm

2.1 Der erste Kontakt mit BASIC

Das Schöne am BASIC des C64 ist, daß Sie an ihm gar nicht vorbeikommen. Sie brauchen das Gerät nur anzuschalten, und schon meldet sich Ihr Commodore mit folgender Meldung:

```
    **** COMMODORE 64 BASIC V2 ****

 64K RAM SYSTEM  38911 BASIC BYTES FREE

READY.
```

Bild 2-1: So meldet sich Ihr C64 nach dem Einschalten

Unter dem R von READY blinkt der sogenannte Cursor, der Ihnen ja jeweils die Stelle anzeigt, an die das Zeichen gesetzt wird, dessen Taste Sie auf der Tastatur betätigen.

Bevor wir damit beginnen, schnell zumindest eine kurze Erklärung der drei angezeigten Zeilen. Denn Sie möchten ja sicherlich wissen, was Ihr C64 Ihnen mitteilt.

In der ersten Zeile wird Ihnen die Version (daher V) des BASIC genannt. In der zweiten Zeile wird die Größe des Arbeitsspeichers angegeben. Es sind beim C64 ca. 64000 BYTES, worunter wir uns der Einfachheit halber etwa Buchstaben vorstellen wollen. Fast die Hälfte davon braucht der Rechner für sich, und die verbleibende Speichermenge wird Ihnen am Schluß als frei (free) angeboten.

Wie man den Cursor auf dem Bildschirm bewegen kann

Und weil der Cursor so schön blinkt und der C64 sich "bereit" (ready) meldet, wollen wir die Gelegenheit gleich nutzen, ein wenig herumzuspielen und uns dabei gemeinsam mit den wichtigsten Tasten vertraut zu machen.

Übrigens: Wenn Sie sich hier schon gut auskennen, überfliegen Sie das Folgende einfach. Sie werden dann schnell merken, wo es sich für Sie lohnt, wieder genauer hinzusehen.

Nun also zum Cursor. Unter der Return-Taste finden Sie zwei Tasten, auf denen jeweils CRSR steht. Außerdem finden Sie insgesamt vier Pfeile für die vier möglichen Richtungen. Wenn Sie die Taste ganz rechts einmal betätigen, geht der Cursor um eine Stelle nach rechts. Wenn Sie die Taste links daneben betätigen, geht es um eine Einheit nach unten. Probieren Sie das jetzt ruhig aus. Achten Sie aber bitte darauf, daß Sie nicht über den unteren Rand hinausgehen. Wenn Sie an den rechten Rand geraten, geht es einfach eine Zeile tiefer links weiter. So wie man auch schreiben würde.

So, haben Sie sich mit den beiden Richtungen vertraut gemacht? Dann probieren Sie es doch einmal mit den beiden anderen Richtungen. Sie bekommen die, wenn Sie gleichzeitig die Shift-Taste links oder rechts gedrückt halten.

Wenn das auch geklappt hat, können wir uns dem Schreiben zuwenden. Wenn Sie statt einer Cursor-Taste einen Buchstaben oder eine Zahl tippen, so bekommen Sie die entsprechenden Zeichen auf den Bildschirm, wobei die Buchstaben allerdings alle groß erscheinen. Wenn Sie zusätzlich noch die Shift-Taste drücken, erscheinen seltsame Grafikzeichen. Mit Hilfe der DEL-Taste ganz oben rechts können Sie das Zeichen, das links neben dem Cursor steht, wieder löschen. Das Zeichen unter dem Cursor und alle rechts davon werden um einen Platz nach links verschoben.

So, jetzt können wir schon eine ganze Menge machen.

Ein kleiner Scherz: Mehr Speicherplatz für den C64

Wenn Sie Lust haben, können Sie sich mit uns zusammen auch noch einen kleinen Scherz erlauben. Am besten schalten Sie Ihren C64 noch einmal kurz aus und dann wieder an, um die frische Einschaltmeldung noch einmal zu erhalten.

Dann gehen Sie mit Hilfe der Cursor-Tasten auf die Zahl in der Mitte der dritten Zeile von oben. Dort werden ja 38911 Bytes als frei angezeigt. Wenn Sie dreimal die Cursor-hoch-Taste (Shift-Taste plus linke Cursor-Taste) und 17mal den Cursor nach rechts bewegt haben, blinkt der kleine Schreibfleck genau auf der ersten Ziffer der Zahl. Sollte Ihnen die Zahl zu gering erscheinen, verändern Sie sie doch einfach, indem Sie fünfmal eine 9 hinschreiben. Und schon wird Ihnen ein mächtiger C64-Speicher mit 99999 freien Einheiten angezeigt. Wenn das nichts ist!!!

Selbst wenn Sie nun die Return-Taste drücken, wird das vom C64 klaglos akzeptiert. Das ist aber eine absolute Ausnahme.

Sobald Sie in die nächste Zeile schreiben:

```
HURRA, ICH HABE JETZT EINEN VIEL GROESSEREN SPEICHER!!!
```

und dann die Return-Taste drücken, erscheint die Meldung

```
?SYNTAX ERROR
```

und darunter wieder das

```
READY.
```

Irgendwie stimmt da etwas nicht!

Nun, es könnte ja sein, daß einem der C64 nur signalisieren will, daß man so einfach an ihm nicht herummanipulieren kann. Also versuchen wir es mit einer freundlichen Begrüßung. Auf die muß er ja wohl positiv reagieren. Und so kommt in die nächste Zeile:

```
HALLO, LIEBER C64!!!
```

Sobald Sie die Return-Taste drücken, erscheint wieder die schon bekannte Meldung.

Das beweist, daß es sich nicht um sachliche Irrtümer wie bei unserer Speichererweiterung, sondern um grundsätzliche Verständigungsschwierigkeiten zwischen unserem C64 und uns handeln muß.

Ein wenig Englisch - kein Problem

Was wir schon ahnen können, ist, daß unser Rechner eine spezielle Sprache spricht, eben BASIC, die anscheinend etwas mit Englisch zu tun hat, wie das ERROR zeigt. Sie können nun natürlich noch das eine oder andere Wort ausprobieren, aber Sie würden ja auch keinem Engländer solange deutsche Worte an den Kopf werfen, bis er Sie versteht. Daher beginnen wir lieber gleich mit dem ersten - und sehr wichtigen - BASIC-Wort. Es lautet PRINT. Wahrscheinlich ahnen Sie jetzt schon, daß alle weiteren Wörter, die wir noch kennenlernen werden, englische bzw. amerikanische Wörter sind. Aber keine Sorge, wir werden das Lernen von Vokabeln stark begrenzen können. PRINT bedeutet übersetzt "drucke", gibt aber entgegen der eigentlichen Bedeutung etwas auf dem Bildschirm aus. Vielleicht knurrt Ihr Drucker jetzt enttäuscht, aber keine Sorge, er bekommt noch genug zu tun.

Geben Sie nun folgende Zeile ein, und drücken Sie anschließend die Return-Taste:

```
PRINT "HALLO"
```

Das Ergebnis ist (direkt darunter):

```
HALLO

READY
```

Sie können sich nun zufrieden auf die Schulter klopfen, denn damit haben Sie den ersten BASIC-Befehl bereits erteilt. Die nächsten Schritte zum Erlernen von BASIC sind nicht schwieriger.

Die Änderung von Befehlseingaben

Beginnen wir am besten sofort mit einer Arbeitserleichterung. Wir wollen den C64 auf den nächsten Seiten noch so allerlei auf den Bildschirm schreiben lassen. Nun wäre es doch viel Arbeit, wenn Sie die ganze Zeile jeweils neu schreiben müßten. Sie können, wie Sie oben gesehen haben, den Cursor mit den vier Pfeiltasten frei auf dem Bildschirm bewegen. Drücken Sie nun viermal <Pfeil nach oben> und zwölfmal <Pfeil nach rechts>. Der Cursor befindet sich nun auf dem zweiten Anführungszeichen. Geben Sie nun einfach ein:

```
MEIN FREUND!" (plus Return-Taste)
```

Wenn Sie sich verschreiben sollten, können Sie mit den Cursor-Tasten und ggf. der DEL-Taste alles problemlos ausbügeln.

Sofort schreibt Ihr C64

```
HALLO MEIN FREUND!
```

auf den Bildschirm. Sie können also auf jede sichtbare Zeile gehen, diese ändern und durch Druck auf die Return-Taste ausführen lassen.

Das Einfügen von Befehlszeilen

Als nächstes wollen wir nichts ändern, sondern etwas einfügen. Und zwar soll vor "FREUND" "BESTER" eingefügt werden.

Das heißt, es sollen sechs Zeichen und ein Leerzeichen eingefügt werden. Um den nötigen freien Platz zu bekommen, können Sie ebenfalls die DEL-Taste verwenden, müssen Sie aber mit der Shift-Taste verbinden. Darum steht auf dieser Taste über dem DEL ein INST, die Abkürzung für INSERT (Einfügen). Mit DEL können Sie ja das Zeichen, auf dem der Cursor steht, um eins nach links verschieben. Das dort stehende Zeichen wird gelöscht. Wenn Sie nun die Shift-Taste gedrückt halten und zusätzlich die DEL/INST-Taste drücken, bekommen Sie Leerzeichen, soviel Sie wollen.

Achtung: Innerhalb des neugeschaffenen Platzes können Sie die Cursor-Tasten nicht normal verwenden. Statt dessen erscheinen eigenartige Zeichen, sogenannte Grafikzeichen. Wenn Ihnen das passiert ist, brauchen Sie nur Shift und Return zu drücken, dann kommen Sie an den Anfang der nächsten Zeile und können sich mit den Cursor-Tasten wieder normal bewegen.

Zurück nun zu unserer Einfügung: In den freien Raum können Sie nun hineinschreiben:

```
BESTER
```

Wenn Sie wieder die Return-Taste betätigen, erscheint auf dem Bildschirm:

```
HALLO MEIN BESTER FREUND!
```

Rechnen mit dem C64

Wir hatten Ihnen ja versprochen, daß wir noch so einiges auf den Bildschirm schreiben würden. Nun wollen wir einmal schauen, ob der C64 auch rechnen kann:

Geben Sie doch einmal ein:

```
3+3
```

Nach dem Drücken der Return-Taste erscheint:

nichts!!!

Fügen wir sicherheitshalber ein Gleichheitszeichen hinzu, ändert sich das niederschmetternde Ergebnis in keiner Weise. Auch wenn wir es mit Anführungsstrichen probieren, kommen wir nicht weiter. Der C64 reagiert zwar, aber leider nur mit einem

```
?SYNTAX ERROR
```

Jetzt fällt uns ein, daß das PRINT, das wir schon kennengelernt haben, vielleicht nötig ist, also:

```
PRINT "3+3 ="
```

Und das Ergebnis ist:

```
3+3 =
READY
```

Na ja, werden Sie wahrscheinlich sagen. Genauso habe ich mir das vorgestellt. Strohdumm das Ding und wiederholt nur, was ich ihm sowieso schon vorgegeben habe. Bevor Ihnen wie uns nun der Geduldsfaden reißt, versuchen wir es einmal ohne Anführungszeichen und auch ohne Gleichheitszeichen:

```
PRINT 3+3
 6

READY
```

Na endlich! So kommen wir weiter. Bevor Sie jetzt Ihren Rechner überglücklich in die Arme schließen und allerlei kleine Rechenaufgaben ausprobieren, wollen wir noch kurz den Unterschied festhalten. Alles, was Sie hinter den PRINT-Befehl in Anführungszeichen schreiben, wird unverändert auf den Bildschirm geschrieben. Soll der C64 etwas ausrechnen, so lassen Sie die Anführungszeichen weg. Auch die anderen Grundrechenarten beherrscht Ihr Rechner souverän:

Die folgenden Befehle führen jeweils zu dem gewünschten Ergebnis:

```
PRINT 3-3 (plus Return-Taste) ergibt 0
PRINT 4*4 (plus Return-Taste) ergibt 16.
```

Auf völlig unerwartete Schwierigkeiten stoßen wir aber bei dem Befehl:

```
PRINT 7:3
```

Da wird uns nämlich als Ergebnis eine 7 angezeigt, und anschließend gibt der C64 auf, indem er den schon bekannten SYNTAX ERROR meldet.

Hoppla! So haben wir aber nicht gewettet. Sollte sich ein so bewährter Rechner wie der C64 vor einer einfachen Division drücken? Keine Sorge! Es funktioniert, nur möchte Ihr Rechner die Aufgabe ein wenig anders gestellt haben. Wenn Sie statt des ":" den Schrägstrich "/" verwenden, funktioniert es nämlich. Also:

```
PRINT 7/3 (plus Return-Taste)
```

ergibt den Wert 2.33333333. Sie sehen, es klappt nicht nur, sondern es wird auch ganz schön genau gearbeitet. Immerhin werden acht Stellen nach dem Komma präsentiert. Aber Halt!!! Was heißt hier Komma? Nun, der Punkt wird Sie wohl kaum verwirren, denn den kennen Sie ja vom Taschenrechner her, wo er auch statt des Kommas verwendet wird. Und genauso macht es auch BASIC.

Kehren wir noch einmal zu unserer letzten kleinen Aufgabe zurück: Vielleicht wächst in Ihnen schon der Verdacht, daß BASIC schneller rechnen kann als Sie, denn für das Ergebnis hätten Sie wahrscheinlich doch etwas länger benötigt. Machen wir einen Test und berechnen die Sekunden eines Tages (60 Sekunden * 60 Minuten * 24 Stunden):

```
PRINT 60*60*24 (Return)
 86400
READY
```

Keine Augenblick zögert der C64 und spuckt prompt das richtige Ergebnis aus.

Bevor wir gleich eine kleine Pause einlegen, möchten wir Sie noch auf eine kleine Besonderheit aufmerksam machen.

Dividieren Sie doch einmal 1 durch 3!

Sie haben jetzt sicherlich nach kurzem Überlegen richtig auf den Bildschirm getippt:

```
PRINT 1/3
```

und als Ergebnis bekommen:

.333333333

Der Punkt als Dezimalzeichen kommt Ihnen nun sicherlich schon bekannt vor. Daß aber eine 0 vor dem Komma nicht dargestellt wird, ist eine Besonderheit, an die Sie sich noch gewöhnen müssen. Vielleicht mögen die Rechner Nullen nicht und verschweigen sie, wo immer es unschädlich ist.

Sie können nun weiter die Rechenfähigkeiten des C64 ausprobieren, sich erst einmal eine kurze Pause gönnen oder mit uns zusammen den nächsten Abschnitt beginnen. Dort erfahren Sie nämlich den Benzinverbrauch Ihres Autos.

Fassen wir zusammen

1. BASIC ist sofort nach dem Starten Ihres C64 da. Es muß nicht erst geladen werden.

2. Man kann den Cursor bei BASIC auf dem gesamten Bildschirm frei bewegen. Dazu benutzt man die Cursor-Tasten.

3. Die Eingabe eines Befehls in einer Zeile und Druck auf die Return-Taste führt dazu, daß BASIC die Zeile sofort ausführt (Direktmodus).

   Dabei sollten Sie darauf achten, daß nichts anderes, auch keine Programmteile o.ä. in der entsprechenden Bildschirmzeile steht. Denn in diesem Falle würde der C64 die ganze Zeile zu verstehen versuchen und wahrscheinlich sein Unverständnis durch ?SYNTAX ERROR kundtun. Wichtig ist die exakte Schreibweise des Befehls. Allerdings kommt es bei den Leerzeichen nicht so darauf an. Man kann die Zahlen und Rechenanweisungen direkt hinter etwa den PRINT-Befehl setzen. Übersichtlicher wird es aber, wenn ein Leerzeichen eingefügt wird (PRINT 17*16).

4. Der zunächst wichtigste BASIC-Befehl ist PRINT. Er gibt Daten auf den Bildschirm (und nicht auf den Drucker, wie das englische Wort zunächst vermuten läßt). Soll der Befehl sich auf ein Wort oder einen Satz beziehen, muß das in Anführungszeichen gesetzt werden: PRINT "Hallo, lieber

C64-Freund" bringt nach Betätigung der Return-Taste genau diese Begrüßung auf den Bildschirm.

5. Den Befehl PRINT können Sie auch verwenden, um sich das Ergebnis von Berechnungen anzeigen zu lassen. Was die Grundrechenarten angeht, so muß beachtet werden, daß das Zeichen für die Division abweichend vom Normalgebrauch der Schrägstrich ist: PRINT 60*60*24/100 ergibt nach Drücken der Return-Taste 864.

   Ähnlich wie beim Taschenrechner wird auch bei BASIC der Punkt statt des Kommas verwendet, um Dezimalstellen abzutrennen: Geben Sie immer 5.5 statt 5,5 ein! Eine Besonderheit ist, daß eine Null vor dem Komma, Pardon: Punkt nicht ausgegeben wird. D.h. PRINT 1/3 ergibt .333333333!

2.2 BASIC bekommt ein Gedächtnis: Das erste Programm

Bei all dem, was wir bis jetzt gemacht haben, wurden BASIC-Befehle genutzt, aber von einem Programm konnte natürlich noch keine Rede sein. Denn nach Erledigung unseres Rechenauftrags war die Arbeit für den C64 in jeder Beziehung erledigt, aus und vorbei.

Im Folgenden werden wir Ihnen zeigen, wie der C64 die BASIC-Befehle gewissermaßen in eine Art Gedächtnis aufnimmt. Er vergißt sie dann erst beim Ausschalten bzw. auf Ihren ausdrücklichen Befehl hin. Diese Gedächtnisleistung ermöglicht es uns überhaupt, Programme, d.h. Abfolgen von Befehlen herzustellen.

Eine kleine Bitte: Sie werden in diesem Abschnitt wertvolle Ergebnisse produzieren, die Sie sicher auf einer Diskette sichern wollen. Zu diesem Zweck sollten wir hier einen kleinen Exkurs einschalten, der Ihnen hilft, eine Diskette für die Aufnahme von Daten vorzubereiten. Sollten Sie eine solche Diskette schon haben (etwa weil Sie die Vorarbeiten aus dem ersten Kapitel ordentlich erledigt haben), würden wir Sie nur bitten, dieser Diskette per Aufkleber den Namen "Arbeitsdiskette1" zu geben. Wir wissen dann immer, von welcher Diskette die Rede ist. Im übri-

gen bitten wir Sie, den folgenden Abschnitt trotzdem zu lesen, da einige grundsätzliche Vorgänge erklärt werden.

Vielleicht nutzen Sie aber auch die Gelegenheit, sich eine zweite oder gar dritte Arbeitsdiskette herzustellen, bei der Sie entsprechend die Nummern hinter dem Wort Arbeitsdiskette entsprechend ändern.

Exkurs: Ohne Vorbereitung geht es nicht - wie man Disketten zur Aufnahme von Daten vorbereitet

Leider ist das insofern ein bißchen schwieriger, als es keinen einfachen kurzen Befehl dafür gibt.

Sie müssen schon eingeben:

```
OPEN 1,8,15,"N:Name,01": CLOSE1
```

Eine Möglichkeit wäre das reine sture Auswendiglernen, aber das ist ziemlich langweilig und auch aufwendig. Besser ist, man versteht die Bestandteile. Dann gewöhnt man sich schnell daran.

Erklärung des Formatierbefehls

Zunächst muß ein sogenannter Befehlskanal aufgemacht werden.

Vorstellung: Es gibt eine Befehlszentrale, zu der man die Tür öffnen muß; der Geheimcode ist 1,8,15,. Das kann man sich sicherlich merken. Denn man merkt sich ja auch eine Telefonnummer oder die Nummer der Euroscheck-Karte, mit der man dem Geldautomaten bis zur Erschöpfung (des Kontos) Geld entlockt.

Anschließend findet sich ein N: Dabei steht N für Neuformatierung. Der folgende Name der Diskette kann beliebig gewählt werden und maximal 16 Zeichen (Buchstaben und Zahlen) lang sein.

Es folgt ein trennendes Komma und ein sogenannter Identifizierungscode, bestehend aus zwei Buchstaben oder Ziffern. Es

empfiehlt sich, die Disketten durchzunumerieren. Und an dieser Stelle könnte man es tun. Erst wenn man bei 99 ist, und das dauert eine Weile, muß man zu Buchstaben übergehen. Man kann aber genausogut die Nummer der Diskette in den Namen aufnehmen und hat dann praktisch unbeschränkten Zahlen-Raum zur Verfügung.

Ganz am Schluß muß die Tür zur Befehlszentrale durch CLOSE wieder sorgfältig verschlossen werden. Das sehen Sie sicher ein. Und damit haben wir alles komplett erklärt.

Wir geben nun ein:

```
OPEN 1,8,15,"N:ARBEITSDISKETTE1,01": CLOSE1 (Return)
```

Das Diskettenlaufwerk produziert einige laute Töne, dann kommt in regelmäßigen Abständen ein leiseres Knacken; nach etwa einer Minute und 20 Sekunden schnurrt es noch einmal behaglich, dann meldet sich der C64 mit READY.

Wie sieht eine Diskette nach dem Formatieren aus?

Das Ergebnis kann man sich nun anschauen, indem man den Befehl ausführen läßt:

```
LOAD "$",8 (Return)
```

Damit lädt sich der C64 das Inhaltsverzeichnis der Diskette in seinen Speicher. Man sieht aber noch nichts. Das ändert sich, sobald man den nächsten Befehl erteilt:

```
LIST (Return)
```

Dann ergibt sich folgendes Bild:

```
OPEN1,8,15,"N:ARBEITSDISKETTE1,01":CLOSE1

READY.
LOAD "$",8

SEARCHING FOR $
LOADING
READY.
LIST

0 "ARBEITSDISKETTE1" 01 2A
664 BLOCKS FREE.
READY.
```

Bild 2-2: Der Formatiervorgang und das Ergebnis

Übrigens: Sie können sich dieses Inhaltsverzeichnis mit LIST (Return) mehrmals nacheinander ansehen. Es ist im Programmspeicher vorhanden.

Erst wenn Sie NEW (Return) eingeben, wird es dort gelöscht, und der LIST-Befehl führt ohne weitere Bildschirmausgaben zu einem READY.

Sicherheit geht vor

Am besten probieren Sie es gleich noch mit einer weiteren Diskette. Zwei sollten Sie auf jeden Fall immer zur Verfügung haben, um alle Programme doppelt abspeichern zu können. Nur so sind Sie hinlänglich vor unangenehmen Überraschungen geschützt. Es ist zumindest theoretisch, aber auch leider praktisch immer mal möglich, daß eine Diskette den Geist aufgibt oder sonst irgendwie das Zeitliche segnet. Bevor wir dann tagelang über die Ursachen oder die Schuldfrage nachgrübeln, ist es besser, wir greifen einfach auf die zweite Diskette zurück und legen eine neue Sicherheitsdiskette an. Die weiteren Disketten könnten in unserem Fall die Namen Arbeitsdiskette2 und Ar-

beitsdiskette3 oder auch eine höhere ID-Nummer (Identifikationsnummer nach dem Namen und dem Komma) bekommen.

Das erste Programm

Vielleicht erinnern Sie sich an unsere Einführung in BASIC. Dort sagten wir, daß man mit BASIC ein Problem vom Computer in kleinen Schritten lösen lassen kann. Was wir aber bisher stets gemacht haben, war: eine Zeile eingeben, Return-Taste drücken, Ergebnis begutachten. Aber wer kann schon die Probleme dieser Welt in einer Zeile beschreiben. Nun gibt es eine Möglichkeit, in BASIC eine Vielzahl von Zeilen nacheinander ausführen lassen. Geben Sie doch einfach folgende Zeilen ein (jeweils mit Return abschließen):

Übrigens: Vorher sollten Sie, falls Sie sich das Inhaltsverzeichnis einer Diskette geladen und angesehen haben, NEW (Return) eingegeben haben. Dann sind Sie sicher, daß nicht noch Reste im Programmspeicher sind, die gleich unsere gemeinsame Beziehung stören, weil bei Ihnen etwas anders funktioniert als bei uns.

Nun also unsere Programmzeilen, die Sie bitte genauso auf den Bildschirm bringen und jeweils mit Return abschließen. Innerhalb einer Zeile können Sie mit Hilfe der Cursor-Tasten sowie der beiden Tasten DEL und INST Tippfehler problemlos berichtigen.

```
10 PRINT "BENZINVERBRAUCH"
20 LITER=150
30 KILOMETER=650
40 VERBRAUCH=LITER/KILOMETER*100
50 PRINT VERBRAUCH
```

Geben Sie nun

```
RUN (Return)
```

ein. Es erscheint folgende Bildschirmausgabe:

```
BENZINVERBRAUCH
 23.0769231

READY
```

Erklärung des Programms

Bevor Sie nun Ihren Wagen in die Werkstatt bringen, wollen wir noch ganz kurz erklären, was da eigentlich abgelaufen ist.

Wenn Sie unter dem READY eine Zeile auf den Bildschirm schreiben, wird diese sofort ausgeführt. Das nennt man in der Computersprache auch gern Direktmodus. Das Gegenstück dazu ist der Programmodus. Dazu muß zu Beginn der Zeile eine Zahl stehen. Diese Zeile wird dann nicht sofort ausgeführt, sondern vom Rechner erst einmal abgelegt. Nach Eingabe von RUN im Direktmodus werden alle so eingegebenen Zeilen (Ihr Programm) nacheinander abgearbeitet.

Unser BASIC-Programm ist jetzt nicht verloren. Sie können es sich leicht anschauen, indem Sie einfach

```
LIST (Return)
```

eingeben. Sofort gibt BASIC die Zeilen 10 bis 50 nacheinander auf dem Bildschirm aus.

2.3 Programme auf Diskette: Speichern, Suchen, Laden, Löschen

Wir können nun anfangen, das Programm .... Aber halt! Diesen Erfolg wollen Sie doch sicherlich festhalten. Legen Sie dazu bitte die leere, aber formatierte Diskette ein, und geben Sie folgende Zeile ein (im Direktmodus, also unter dem READY ohne Zeilennummer):

```
SAVE "BENZINVERBRAUCH",8 (plus Return-Taste)
```

Die Lampe der Floppy leuchtet kurz auf, und auf dem Bildschirm erscheint die Meldung:

```
SAVING BENZINVERBRAUCH
```

Während das abläuft, wollen wir Ihnen kurz den Befehl erklären. SAVE ist ein BASIC-Befehl und heißt ganz einfach "Speichere!". Das Besondere beim C64 ist nun, daß er die Angabe verlangt, wohin gespeichert werden soll. Im Falle eines Diskettenlaufwerks ist 8 die entsprechende Nummer. Und sie muß immer hinter Lade- und Speicherbefehle gesetzt werden, durch ein Komma vom Programmnamen getrennt.

Sobald dann wieder READY erscheint, ist Ihr erstes Programm gespeichert. Oder glauben Sie uns etwa nicht. Wir haben Sie doch bisher noch nie enttäuscht. Na gut, Vertrauen ist gut - Kontrolle ist besser. Mit der Befehlszeile:

```
LOAD "$",8 (Return)
```

und anschließendem LIST können Sie sich das aktuelle Inhaltsverzeichnis anschauen. Wenn Sie wirklich eine leere Diskette eingelegt haben, erscheint folgende Bildschirmausgabe:

```
LOAD "$",8

SEARCHING FOR $
LOADING
READY.
LIST

0 "ARBEITSDISKETTE1" 01 2A
1     "BENZINVERBRAUCH"  PRG
663 BLOCKS FREE.
READY.
```

Bild 2-3: Inhaltsverzeichnis - Ein Programm glücklich abgespeichert

Woraus wir zweierlei lernen können:

1. Können Sie uns vertrauen und
2. können und dürfen Dateinamen nicht länger als 16 Zeichen sein.

Preisfrage: Wie können Sie nun den echten Benzinverbrauch Ihres Wagens bestimmen? Richtig! Sie lassen sich die Zeilen Ihres Programms mit LIST zeigen - Halt Stop! Sobald Sie LIST (Return) eingeben, bekommen Sie immer wieder Ihr Inhaltsverzeichnis der Diskette. Es wird ja wie ein Programm behandelt und steckt schön im Programmspeicher Ihres Rechners. Nun, kein Problem. So können wir gleich sehen, wie man ein Programm wieder lädt, denn wir haben es ja abgespeichert.

Sie sollten zunächst wieder NEW (Return) eingeben und anschließend das Programm wie folgt wieder laden:

```
LOAD "BENZINVERBRAUCH",8 (Return)
```

Mit LIST (Return) lassen Sie es sich wieder anzeigen und tragen in den Zeilen 20 und 30 die richtigen Werte für Ihr Auto ein.

Abänderung von Werten in einem Programm

Nehmen wir an, Sie sind 300 Kilometer gefahren, und Sie mußten 18 Liter tanken. Bewegen Sie den Cursor auf die Zeile mit der Nummer 20, und überschreiben Sie die 150 durch 18. Ebenso wird die 650 durch 300 überschrieben. Sie müssen nur zwei Dinge beachten:

1. Eine Zeile wird nur dann im Programm geändert, wenn Sie nach der Änderung die Return-Taste drücken. Sollten Sie also die Änderung nicht wollen, so verlassen Sie einfach die Zeile ohne Druck auf diese Taste.

2. Schreiben Sie die Befehle im Direktmodus (also LIST oder RUN) möglichst in eine leere Zeile. Das heißt: Wenn Sie etwas geändert und die Änderung durch Betätigung der Return-Taste abgeschlossen haben, gehen Sie am besten mit dem Cursor unter das Programmende, bevor Sie z.B. RUN (plus Return-Taste) anordnen. Sonst nimmt Ihr C64

alles ernst, was neben RUN noch in der Bildschirmzeile steht. Und das führt dann zu einem ?SYNTAX ERROR. Sollte sogar eine Zahl zu Beginn der Zeile stehen, so reagiert Ihr C64 gar nicht, weil er sich dadurch im Programm-Modus befindet. Dafür haben Sie hinterher aber eine Programmzeile, die spätestens nach RUN (plus Return-Taste) zu einem ?SYNTAX ERROR führt. Und das heißt ja nichts anderes, als daß Ihr auf ordentliche BASIC-Befehle fixierter C64 Ihre Anweisungen nicht versteht. Und Sie meinten ja auch eigentlich etwas ganz anderes.

Um nun die Änderung zu kontrollieren, bewegen Sie den Cursor also in eine leere Zeile und geben LIST ein. Na, sind die Änderungen im Programm vorhanden? Gut, dann können Sie jetzt RUN eingeben, somit Ihr Programm ausführen lassen und sich das Ergebnis anschauen:

```
RUN (plus Return-Taste)
```

Und es erscheint:

```
Benzinverbrauch
 6
READY
```

Gut, werden Sie sagen. Feine Sache. Nach jedem Tanken berechne ich nun den Benzinverbrauch des Wagens, und wenn er zu viel säuft, kommt er weg - schon hat sich die Ausgabe für das Buch gelohnt. Und wie Sie Ihr Programm immer wieder von Diskette schnell und sicher laden können, haben wir ja oben gesehen. Zur Wiederholung führen wir den Befehl hier noch einmal auf:

```
LOAD "BENZINVERBRAUCH",8 (Return)
```

Schon haben wir einen zufriedenen Autobesitzer (hoffentlich).

Wie man ein Programm auf Diskette löscht

An dieser Stelle wollen wir Sie wenigstens darauf hinweisen, daß Sie Programme auch mit Hilfe eines BASIC-Befehls wieder löschen können. Zu diesem Zweck sollten Sie zunächst natürlich ein Programm erzeugen, das Sie sofort wieder löschen können:

Das geht ganz einfach, indem wir das bereits fertige Programm BENZINVERBRAUCH nehmen, das ja noch im Speicher ist oder schnell geladen wird. Wir speichern es jetzt noch einmal ab unter:

```
SAVE "TEST",8 (Return)
```

Wie können wir dieses (überflüssige) Programm nun löschen?

Auch hier müssen wir wieder auf die Befehlszentrale zurückgreifen, die Sie ja schon vom Formatiervorgang her kennen. Nun, Sie erinnern sich sicherlich noch an die Code-Nummer. Der komplette Befehl für das Formatieren hieß:

```
OPEN 1,8,15,"N:BASIC,01":CLOSE1 (plus Return-Taste)
```

Der entsprechende Löschbefehl heißt nun:

```
OPEN1,8,15,"S:TEST":CLOSE1
```

Der Buchstabe S steht für das englische Wort "scratch", was soviel wie "löschen" heißt.

Ein erneuter Blick ins Inhaltsverzeichnis zeigt uns, daß unser Programm TEST erfolgreich gelöscht worden ist. Sie wissen doch noch, wie das geht: LOAD "$",8 und LIST.

2.4 Der Computer stellt Fragen: INPUT

Kehren wir zu unserem Benzin-Verbrauchsprogramm zurück. Wir hatten davon gesprochen, daß Sie nach jedem Tanken die neuen, aktuellen Werte ins Programm eintragen können. Auf die Dauer ist es aber lästig, jedesmal das Programm "listen" und dann die richtigen Werte eintragen zu müssen. Deshalb wollen wir Ihnen jetzt eine komfortablere Möglichkeit vorstellen.

Zunächst sollten Sie aber mit NEW (Return) Ihren Programmspeicher frei machen.

Wenn es Sie übrigens stört, daß Sie noch so viel Programm- und Eingabereste auf dem Bildschirm stehen haben, können Sie den ebenfalls frei machen. Es gibt dafür eine spezielle Taste. Sie liegt links neben der DEL/INST-Taste und trägt die Buchstaben CLR als Abkürzung für Clear (klarer Bildschirm). Sobald Sie sie betätigen (zusammen mit einer Shift-Taste, da das CLR oben auf der Taste steht), wird der gesamte Bildschirm frei gemacht, und der Cursor ist ganz oben links in der Ecke, bereit zu neuen Taten.

Jetzt sollten Sie zusammen mit uns folgendes Programm ausprobieren (bitte wieder jede Zeile mit Return abschließen!)

```
10 INPUT LITER
20 PRINT LITER
30.INPUT KILOMETER
40 PRINT KILOMETER
```

Wenn Sie das Programm mit RUN starten, erscheint nacheinander zweimal ein Fragezeichen, und die von Ihnen eingegebenen Zahlen werden anschließend sofort wieder auf den Bildschirm geschrieben. Das wollen wir doch gleich in unserem Programm BENZINVERBRAUCH einsetzen. Zuerst löschen wir das jetzige Programm wieder mit NEW (Return) und laden unser Verbrauchs-Programm mit

```
LOAD "BENZINVERBRAUCH",8 (Return)
```

von der Diskette. Zum Ändern listen wir es mit LIST auf. Anschließend werden zwei Zeilen folgendermaßen geändert:

```
20 INPUT LITER
30 INPUT KILOMETER
```

Dazu gehen wir mit dem Cursor in die entsprechende Zeile, überschreiben den bisherigen Inhalt mit den neuen Zeichen und ändern die Zeile endgültig mit der Return-Taste. Schauen Sie sich das geänderte Programm ruhig noch einmal mit LIST an. Es sollte jetzt so aussehen:

```
10 PRINT "BENZINVERBRAUCH"
20 INPUT LITER
30 INPUT KILOMETER
40 VERBRAUCH=LITER/KILOMETER*100
50 PRINT VERBRAUCH
```

Starten Sie das geänderte Programm mit RUN. Hinter dem ersten Fragezeichen geben Sie 40 und hinter dem zweiten 600 ein. Anschließend gibt BASIC 6.66666667 als Verbrauch aus. Sie können das Programm nun einfach erneut (mit RUN plus Return) starten, andere Werte eingeben und sich den geänderten Verbrauch anzeigen lassen. Wenn Sie dann den Verbrauch aller Autos in Ihrer Nachbarschaft berechnet und verglichen haben, speichern Sie Ihr Werk am besten mit

```
SAVE "INPUT",8 (Return)
```

auf der Diskette ab. Wir wollen nämlich noch eine letzte Verbesserung am Programm vornehmen.

Wie man Computerfragen aussagekräftig macht

Vielleicht ist es Ihnen nämlich auch so gegangen wie uns. Nach dem dritten Starten des Programms erschien das Fragezeichen, wir wußten aber nicht mehr genau, ob wir jetzt erst die Liter oder die Kilometer eingeben mußten. So ein Fragezeichen sagt nämlich relativ wenig über den einzugebenden Wert aus. Man müßte noch irgendeinen Hinweis auf dem Bildschirm haben, was an dieser Stelle einzugeben ist. Klar, werden Sie sagen. Das ist

doch keine Schwierigkeit. Wir schreiben einfach mit PRINT vorher auf den Bildschirm, was eingegeben werden soll. Gut, probieren wir das doch gleich einmal aus.

Jetzt erweist es sich als günstig, daß wir Zehner-Abstände zwischen den Programmzeilen gelassen haben. Wir können jetzt nämlich problemlos zwei Zeilen einfügen. Sicherheitshalber nehmen wir die Hälfte zwischen den Zeilen, dann haben wir auch für spätere Programmänderungen noch Platz.

Geben Sie also die beiden folgenden Zeilen ein (Abschluß mit Return nicht vergessen!)

```
15 PRINT "BITTE LITER EINGEBEN"
25 PRINT "BITTE KILOMETER EINGEBEN"
```

Nun können Sie mit LIST das Ergebnis anschauen:

```
10 PRINT "BENZINVERBRAUCH"
15 PRINT "BITTE LITER EINGEBEN"
20 INPUT LITER
25 PRINT "BITTE KILOMETER EINGEBEN"
30 INPUT KILOMETER
40 VERBRAUCH=LITER/KILOMETER*100
50 PRINT VERBRAUCH
```

Wie erwartet, hat BASIC die beiden Zeilen an der gewünschten Stelle eingefügt. Starten Sie das Programm nun mit RUN. Zusammen mit Ihren Eingaben entsteht folgender Bildschirm:

```
BENZINVERBRAUCH
BITTE LITER EINGEBEN
? 40
BITTE KILOMETER EINGEBEN
? 600
 6.66666667
READY
```

Nun können wir uns bei der Eingabe hinter dem Fragezeichen nicht mehr vertun. Wenn wir zum Schluß noch eine Zeile vor Zeile 10 setzen, in der ein Hinweis zu der nachfolgenden, vom C64 ausgegebenen Zahl auf den Bildschirm geschrieben wird, ist unser erstes Programm endgültig fertig. Sicherlich sind Sie jetzt

schon so sicher im Umgang mit Zeilennummern, daß Ihnen die Zeile

```
45 PRINT "DER VERBRAUCH IST"
```

keine Schwierigkeiten mehr macht. Vergessen Sie bitte nicht, das Programm in seiner endgültigen Fassung abzuspeichern mit:

```
SAVE "BENZIN MIT INPUT",8 (Return)
```

Und weil es so schön ist, drucken wir es hier noch einmal komplett ab!

```
10 PRINT "BENZINVERBRAUCH"
15 PRINT "BITTE LITER EINGEBEN"
20 INPUT LITER
25 PRINT "BITTE KILOMETER EINGEBEN"
30 INPUT KILOMETER
40 VERBRAUCH=LITER/KILOMETER*100
45 PRINT "DER VERBRAUCH IST"
50 PRINT VERBRAUCH
```

2.5 Programme schwarz auf weiß: Das Ausdrucken

Vielleicht möchten Sie es auch gerne ausgedruckt haben. Das ist kein Problem, erfordert allerdings ein wenig Tipp-Arbeit auf dem Bildschirm. Um so schöner und arbeitssparender ist dann aber das Ergebnis.

Wir gehen davon aus, daß Sie unser Programm zur Berechnung des Benzinverbrauchs in seiner letzten Fassung geladen haben. Sonst sollten Sie das mit

```
LOAD"BENZIN MIT INPUT",8
```

schnell erledigen.

Als nächstes müssen wir wieder eine Tür aufmachen, diesmal die Tür zum Drucker: Der hat natürlich eine andere Adresse als unsere schon bekannte Befehlszentrale:

```
OPEN1,4 (Return)
```

Dann brauchen wir einen Befehl, der alles, was normalerweise auf den Bildschirm geschickt würde, auf den Drucker umleitet:

```
CMD1 (Return)
```

Daß das funktioniert, merken wir bereits daran, daß das folgende READY nicht mehr auf den Bildschirm, sondern bereits auf dem Drucker ausgegeben wird.

Und nun brauchen wir nur noch den ganz normalen LIST-Befehl, also:

```
LIST (Return)
```

Anschließend müssen die beiden Befehle wieder rückgängig gemacht werden, das geschieht durch:

```
PRINT#1 (Return)
```

und

```
CLOSE1 (Return).
```

Anschließend können wir ganz normal am Bildschirm weiterarbeiten.

Das Ausdrucken von Programmen

Also noch einmal die Befehlsabfolge für das Ausdrucken von Programmzeilen:

```
OPEN1,4 (Return)
CMD1 (Return)
LIST (Return)
PRINT#1 (Return)
CLOSE1 (Return)
```

Und schon haben wir eine tolle Möglichkeit, unsere Programme schwarz auf weiß vor uns zu haben. Zum einen werden die Augen nach stundenlanger Arbeit am Bildschirm geschont, zum anderen hat man natürlich auf dem Papier einen schnelleren Überblick.

Schon bald werden Sie auch eine Möglichkeit kennenlernen, wie man die Ergebnisse eines Programms ebenfalls ausdrucken kann und dann schwarz auf weiß besitzt.

2.6 Zusammenfassung

Sie haben nun einen ersten, kurzen Einblick in die Arbeit mit BASIC erhalten. Bevor wir uns im nächsten Abschnitt eingehender mit einigen Fähigkeiten auseinandersetzen, wollen wir die Erfahrungen und Ergebnisse noch einmal kurz festhalten:

Was wir schon wußten;

1. BASIC ist nach dem Einschalten sofort da, und es kann losgehen..
2. Man kann den Cursor bei BASIC auf dem gesamten Bildschirm frei bewegen. Dazu benutzt man die Cursor-Tasten.
3. Die Eingabe einer Zeile und Druck auf die Return-Taste führt dazu, daß BASIC die Zeile sofort ausführt (Direktmodus).

Dabei sollten Sie darauf achten, daß nichts anderes, auch keine Programmteile o.ä., in der entsprechenden Bildschirmzeile steht. Denn in diesem Falle würde der C64 die ganze Zeile zu verstehen versuchen und wahrscheinlich sein Unverständnis durch ?SYNTAX ERROR kundtun. Der C64 kennt seine BASIC-Befehle so gut, daß er sie auch in den seltsamsten Verkleidungen erkennt, d.h. Sie können ihm sowohl den Befehl PRINT LITER als auch den Befehl PRINTLITER erteilen. In beiden Fällen weiß er, was er tun soll, nämlich das auf den Bildschirm schreiben, was er unter LITER versteht. In unserem Beispiel stand LITER ja für die getankten Liter.

Außerdem haben wir kennengelernt:

1. Steht zu Beginn der Zeile eine Zahl, so wird die Zeile als Programm oder Teil eines Programms verstanden und an der richtigen Stelle (an Hand der Zeilennummer) abgelegt (Programm-Modus).

2. Sie können eine Programmzeile leicht ändern, indem Sie mit dem Cursor an die entsprechende Stelle gehen, dort die Änderung vornehmen und das Ganze durch Druck auf die Return-Taste (nicht vergessen!!!) auch Ihrem C64 klarmachen. Anschließend ist die Änderung im Rechnerspeicher.

3. Folgende Wörter und Befehle der Sprache BASIC kennen Sie nun bereits (im Direktmodus bitte nie den Abschluß durch Druck auf die Return-Taste vergessen):

   PRINT gibt etwas auf dem Bildschirm aus. Setzt man das Auszugebende in Anführungsstriche, so wird es unverändert ausgegeben, ohne die Anführungsstriche werden Berechnungen oder Werte eines Platzhalters (Verbrauch) ausgegeben.

   Durch die Benutzung der Zeichen für die Grundrechenarten kann addiert (+), subtrahiert (-), multipliziert (*) und dividiert (/) werden.

   LIST: Zeigt die bisher eingegebenen Programmzeilen nach Zeilennummern geordnet auf dem Bildschirm. Dabei kann die Zeile mit neuem Inhalt überschrieben werden und wird nach dem Drücken der Return-Taste endgültig geändert.

   NEW: Löscht den Programmspeicher und sollte zu Beginn jedes neuen Programms eingegeben werden.

   LOAD "NAME",8 lädt das Programm "NAME" von der Diskette.

   SAVE "NAME",8 speichert das Programm "NAME" auf Diskette.

   LOAD "$",8 zeigt das aktuelle Inhaltsverzeichnis der Diskette.

   Mit OPEN1,8,15,"S:NAME":CLOSE1 können Sie ein Programm mit dem Namen NAME wieder löschen.

   INPUT LITER (als Befehl in einem Programm): Der hinter dem erscheinenden Fragezeichen eingegebene Wert wird von BASIC unter dem Namen "LITER" gespeichert.

Ausdrucken kann man die Zeilen eines Programms mit der folgenden Befehlsfolge:

```
OPEN1,4 (Return)
CMD1 (Return)
LIST (Return)
PRINT#1 (Return)
CLOSE1 (Return)
```

2.7 BASIC als erste Hilfe für den Alltag

Bevor wir tiefer in die Materie einsteigen, möchten wir Sie auf einige Möglichkeiten hinweisen, wie Sie zu diesem Zeitpunkt schon BASIC nutzen können. Sicher, das mit der Benzinverbrauchsberechnung war ja ganz schön und auch hilfreich, aber in der Praxis kommen noch viel mehr Anwendungen vor, für die Sie bisher Ihren C64 nicht einsetzen konnten.

Nehmen wir einmal an, Sie haben die Einkäufe für eine Party hinter sich gebracht. Die Kosten sollen glücklicherweise umgelegt werden, so daß Ihre Kasse nicht übermäßig beansprucht wird. Jetzt müssen Sie zunächst einmal feststellen, wieviel Sie insgesamt ausgegeben haben. Sie haben sicherlich für eine solche umfangreichere Addition schon seit längerem nicht mehr Papier und Bleistift, sondern einen Taschenrechner verwendet. Das funktionierte auch ganz gut, nur: Wenn Sie zur Kontrolle rückwärts rechneten, kam häufig ein anderes Ergebnis heraus, und der Fehler ließ sich nur finden, wenn Sie noch ein- oder zweimal alles eingaben, bis Ihnen das am häufigsten auftauchende Ergebnis das richtige zu sein schien.

In BASIC geht das viel einfacher: Sie geben einfach Ihre Teilausgaben als Programmzeilen aus. Also zum Beispiel:

```
10 AUSGABEN = 18.60+8.5+12.5+4+5.50+12+2.20+5.70+17
20 PRINT AUSGABEN
```

Sobald Sie RUN (Return) eingegeben haben, haben Sie das richtige Ergebnis auf dem Bildschirm. Und wenn es Ihnen etwas seltsam vorkommt, etwa weil es sich um einige Zehntausend Mark handelt, können Sie durch einfache Kontrolle der Bildschirmzeile feststellen, wo Sie das Komma bzw. in BASIC den Punkt falsch gesetzt haben. Eine kleine Korrektur in der Zeile,

abgeschlossen durch die Return-Taste, und schon haben Sie das gewünschte Ergebnis richtig auf dem Bildschirm.

Sollten Sie mehr Ausgaben haben, als in eine Zeile passen (dann wäre es entweder schon eine Riesenfete, oder aber Sie haben sehr viele Klein-Einkäufe getätigt), fügen Sie einfach eine Zeile 15 ein. Bei der müssen Sie natürlich das Zwischenergebnis aus Zeile 10 wie folgt aufnehmen:

```
15 AUSGABEN = AUSGABEN+80.60+50+75.5+90+6.8
```

Ihr gesamtes kleines, aber sehr praktisches Programm sieht dann wie folgt aus:

```
10 AUSGABEN = 18.60+8.5+12.5+4+5.50+12+2.20+5.70+17
15 AUSGABEN = AUSGABEN+80.60+50+75.5+90+6.8
20 PRINT AUSGABEN
```

Und wenn Sie nun wissen wollen, wieviel Sie jedem Ihrer 12 Teilnehmer abknöpfen müssen, kein Problem. Sie fügen einfach eine Zeile 30 ein, die so aussehen könnte:

```
30 PRINT AUSGABEN/12
```

Und schon haben Sie den genauen Wert. Sollten übrigens noch unangemeldet zwei weitere Gäste kommen und mit ans "Eingemachte" gehen, ebenfalls kein Problem. Denn Sie haben natürlich das Programm gespeichert und ersetzen einfach in der Zeile 30 die 12 durch eine 14 und haben den aktuellen Wert.

Ganz schön praktisch, nicht wahr?

Das selbe Verfahren können Sie anwenden, wenn Sie etwa Ihre durchschnittlichen Telefonkosten im Monat berechnen wollen.

So etwa könnte ein entsprechendes Programm aussehen.

```
10 TELEF = 50+56.7+48+58.6+70+124+143+40+38+35.7+32.50+34
20 DURCHSCNITT = TELEF/12
30 PRINT "GESAMTTELEFONKOSTEN ="
40 PRINT TELEF
```

```
50 PRINT "DURCHSCHNITTLICHE TELEFONKOSTEN PRO MONAT"
60 PRINT DURCHSCHNITT
```

Die hohen Zahlen in der Mitte entstanden wahrscheinlich dadurch, daß Sie zwar einen C64, aber noch kein Einsteigerbuch zu ihm und seinen Möglichkeiten gekauft hatten. Und all Ihre Bekannten waren am Telefon genauso ratlos wie Sie selbst. Nun, das ist vorbei.

Was die Bezeichnungen angeht, so steht TELEF für Telefonkosten. Aus bestimmten Gründen, auf die wir später noch genauer eingehen werden, hält der C64 nichts von dem Wort TELEFON.

Und zwar liegt das an den beiden Buchstaben ON. Die versteht BASIC nämlich als einen Befehl, und dann kann es mit dem Rest (TELEF) nichts mehr anfangen. Aber davon später mehr.

Das soll fürs erste reichen. Sie werden sicherlich noch einige andere Berechnungen finden, die für Sie persönlich besonders wichtig sind und Tag für Tag anfallen. Probieren Sie das mal selbst aus, wobei Sie alle Grundrechenarten verwenden können.

Wenn Sie ähnlich wie AUSGABEN, LITER und BENZIN dabei entsprechende Begriffe verwenden wollen, nehmen Sie möglichst kurze, eventuell auch Abkürzungen, und achten Sie darauf, daß in den Namen keine BASIC-Befehle vorkommen. Dazu gehören auch die kurzen Befehle: OR, FN, IF, TI, und ST. Außerdem sollen sich unterschiedliche Begriffe unbedingt in den ersten beiden Buchstaben unterscheiden.

Aber damit sind wir bereits mitten im nächsten Kapitel. Wenn Sie also mehr darüber wissen wollen, warum diese Hinweise wichtig sind, sollten Sie eine kurze Pause einlegen, und dann nichts wie hinein!

3. BASIC an einem Wochenende

Na, haben Sie sich eine kleine Pause gegönnt und sich etwas erholt? Fein, dann können wir Sie ja noch etwas weiter in die Möglichkeiten des Programmierens einführen.

Gleich zu Beginn müssen wir Ihnen gestehen, daß wir uns im letzten Abschnitt um eine ganz wichtige Thematik herumgemogelt haben. Wir haben inständig gehofft, daß Sie sich eine ganz bestimmte Frage nicht stellen, weil wir beim ersten Kennenlernen von BASIC nicht auf diesen Punkt eingehen wollten.

3.1 Schubladen und Platzhalter: Variablen für Zahlen in BASIC

Diese Frage könnte man in etwa so stellen: Wie kommt mein C64 dazu, etwas von LITER, KILOMETER und VERBRAUCH zu verstehen? Daß er PRINT, INPUT und LIST versteht, ist ja noch einigermaßen einleuchtend, denn diese Wörter hat man ihm eben beigebracht. Aber an Liter und Kilometer können die Programmierer von BASIC doch auf keinen Fall gedacht haben.

In der Zusammenfassung am Ende des letzten Kapitels haben wir beim PRINT-Befehl schon den Begriff Platzhalter gebraucht. Gebräuchlicher ist aber der Begriff Variable. Im Programm Benzinverbrauch sind LITER, KILOMETER und VERBRAUCH Variablen. Solche Platzhalter nehmen in Programmen eine ganz wichtige Stellung ein, weil man dadurch mit verschiedenen, sich ändernden, variablen Werten arbeiten kann. Vielleicht sind Ihnen im Programm Benzinverbrauch diese Variablen nur deshalb nicht ungewöhnlich vorgekommen, weil wir ihnen Namen gegeben haben, die sofort einleuchten und die deshalb gar nicht aufgefallen sind. Genausogut hätten wir das Programm aber auch folgendermaßen schreiben können:

```
1 PRINT "BENZINVERBRAUCH"
2 INPUT A
3 INPUT B
4 C = A / B * 100
5 PRINT C
```

Verstehen Sie noch etwas? Wir auch nicht. Aber probieren Sie es ruhig aus, dieses Programm liefert dieselben richtigen Ergebnisse. Nur einleuchtend ist, daß die richtige Wahl der Variablennamen ein Programm viel verständlicher macht als beliebige Zeichen. Es gibt allerdings einige Einschränkungen bei der Wahl der Variablennamen:

Was man bei der Wahl von Variablennamen beachten muß

1. Ihre Gesamtlänge kann zwar fast über zwei Bildschirmzeilen gehen, aber es dürfen keine Leerzeichen auftreten. Außerdem sind solch lange Namen natürlich in der Praxis unhandlich.

2. Außerdem: Unterschieden werden sowieso nur die ersten beiden Zeichen, d.h. KILOMETER und KILOGRAMM wären für den C64 identisch, was zu lustigen Verwechslungen etwa bei der Berechnung des Idealgewichts führen könnte.

3. Außerdem: In Variablennamen dürfen sich keine BASIC-Befehle verstecken: BRUNNEN etwa ist unzulässig, weil es RUN enthält.

   Die wichtigsten von den besonders gefährlichen Zwei-Buchstaben-Befehlen sind:

   ```
   OR, FN, IF, TI, TI$, ST
   ```

   Auch unter diesem Gesichtspunkt empfiehlt es sich, kurze Variablennamen zu nehmen!

4. Ziffern dürfen verwendet werden, aber die Zahl darf erst an zweiter Stelle auftauchen: 6A etwa ist unzulässig.

5. Erwähnt worden ist schon, daß Variablennamen nicht durch Leerzeichen unterbrochen werden dürfen.

Wie man mit Variablen umgeht

Das Nützliche an diesen Variablen ist, daß man sie in allen Rechenvorschriften genauso wie Zahlen einsetzen kann. Das haben wir ja auch schon in unserem Programm Benzinverbrauch in Zeile 20 ausgenutzt. Mit diesem Wissen dürften Ihnen eigentlich Zeilen wie

```
20 LITER = 150
40 VERBRAUCH = LITER / KILOMETER * 100
```

kaum noch Schwierigkeiten bereiten.

Aber wie war das bei dem Einsatz unseres C64s als komfortabler Taschenrechner am Ende des letzten Kapitels: Da tauchte doch eine Zeile auf:

```
15 AUSGABEN = AUSGABEN + 20 +...
```

und dann kamen noch weitere Zahlen.

Wir sehen schon, wie die Mathematiker unter Ihnen sich den Schweiß von der Stirn wischen. Das ist keine Gleichung, keine Ungleichung, das ist ein Unding. Kürzt man nämlich auf beiden Seiten die AUSGABEN aus der Gleichung, so steht dort glatt "0= 20". Trotzdem ist diese Zeile für BASIC kein Problem, wie Sie mit folgendem kleinen Programm leicht feststellen können:

```
10 AUSGABEN = 50
20 AUSGABEN = AUSGABEN + 20
30 PRINT AUSGABEN
```

Wenn Sie dieses Programm abgetippt haben (vorher bitte mit dem Befehl NEW plus Return-Taste den Programmspeicher leeren!) und mit RUN starten, erhalten Sie als Ergebnis 70. In Zeile 10 hat BASIC der Variablen AUSGABEN den Wert 50 zugewiesen. In der nächsten Zeile wurde dann der Variablen AUSGABEN der Wert (AUSGABEN + 20), also 50 + 20 = 70 zugewiesen. Variablen können also nicht nur überall an Stelle von Zahlen eingesetzt werden. Sie stellen auch eine Art Schubladen dar, in denen beliebige Werte, vor allem auch Ergebnisse von Be-

rechnungen abgelegt werden können. Das Gleichheitszeichen entspricht in diesem Zusammenhang also eher einer Zuweisung.

Übrigens gibt es für das Gleichheitszeichen in diesem Sinne auch den BASIC-Befehl LET. Es heißt dann also:

```
20 LET AUSGABEN = AUSGABEN + 20.
```

Damit ist der Sachverhalt dann klar! Aber wir wollen doch lieber bei dem kürzeren Gleichheitszeichen bleiben.

3.2 Variablen können auch Wörter enthalten: Stringvariablen

Leider haben diese Variablen, die Sie bisher kennengelernt haben, neben den vielen Vorteilen einen großen Nachteil. Sie können wirklich nur Zahlen aufnehmen. Es ist nicht möglich, einer solchen Variablen beispielsweise einen Vornamen zuzuweisen. Das wollen wir gleich einmal ausprobieren:

Übrigens: Wir haben es zunächst mit den Variablennamen VORNAME statt NAME probiert. Ihnen wollen wir diesen Reinfall ersparen, denn natürlich wird VORNAME vom C64 als eine unsinnige Verbindung von irgendwelchen Buchstaben mit dem Befehl OR aufgefaßt und nicht akzeptiert.

```
NAME = PETER
```

Gleichmütig meldet BASIC sein READY. Aha, also haben sich die Autoren diesmal vertan. Wenn diese Zeile nicht erlaubt wäre, hätte es doch diesen ominösen ?SYNTAX ERROR gegeben. Also gut, schreiben wir ein kleines Programm. Vorher am besten NEW eingeben, um auf jeden Fall einen leeren Programmspeicher zu haben!

```
10 NAME = PETER
20 PRINT NAME
```

Nach dem Start mit

```
RUN
```

erhalten wir:

```
0
READY
```

Nun, selbst wenn Sie Peter nicht leiden mögen, das geht entschieden zu weit, daß man ihn jetzt zur Null macht. Bevor Sie jetzt die Namen aller Ihrer Freunde ausprobieren, wollen wir es Ihnen lieber gleich verraten. Obwohl PETER sich sehr nach einem Namen anhört, ist er für BASIC genauso eine Variable wie LITER, KILOMETER oder ein allgemeines A. Und jede Variable, die noch keinen besonderen Wert hat, wird vom C64 automatisch auf 0 gesetzt. Und daher ist in Zeile 10 PETER = 0, und dann natürlich auch NAME.

Versuchen wir es doch einmal wie am Anfang mit Anführungszeichen:

```
10 NAME = "PETER"
```

Das anschließende RUN beschert eine neue Meldung von BASIC und läßt zum Wörterbuch greifen: "TYPE MISMATCH ERROR IN 10", wörtlich übersetzt: "Typ! Du bist gematscht und irrst Dich in Zeile 10".

BASIC hat hier pfiffigerweise erkannt, daß wir versucht haben, einer Variablen, die nur Zahlen aufnehmen kann, ein Wort zuzuweisen. Trotzdem läßt der Hinweis auf den Typ schon ahnen, daß es Variablen gibt, die so etwas durchaus können, nur NAME eben nicht.

Das Dollarzeichen als Kennzeichen der Stringvariablen

Ändern Sie doch die Zeile 10 einfach einmal in folgende Zeile um:

```
10 NAME$ = "Peter"
20 PRINT NAME$
```

Das anschließende RUN bringt nun endlich den gewünschten Namen auf den Bildschirm. Aha! Der entscheidende Unterschied

im Variablennamen ist also das Dollarzeichen am Ende. Dadurch weiß BASIC also, daß dieser Variablen Namen, andere Wörter oder ganz allgemein Zeichengruppen zugewiesen werden. Im Gegensatz zu den Variablen, die mit Zahlen gefüllt werden (numerischen Variablen), spricht man hier auch von alphanumerischen Variablen. Das sind einfach Platzhalter, die für beliebige Alphas und seine Kollegen, die anderen Buchstaben stehen können. Übrigens: Wenn Sie sich einmal mit dem Anhängsel $ für den Typ Alphanumerische Variablen entschieden haben, müssen Sie auch dabei bleiben. Dies können Sie leicht dadurch feststellen, daß Sie in der Zeile 10 PETER durch eine Zahl, etwa die 1 ersetzen. Sofort bekommen Sie wieder die schon bekannte Fehlermeldung.

Man nennt Variablen mit einem Dollarzeichen am Schluß übrigens Stringvariablen. Sie können also fast beliebige Zeichen zwischen zwei Anführungsstrichen aufnehmen, nur rechnen kann man mit Ihnen nicht mehr. Daß man sie aber trotzdem gut gebrauchen kann, soll das folgende kleine Programm zeigen:

```
10 PRINT "BITTE VORNAME EINGEBEN"
20 INPUT VN$
30 PRINT "BITTE NACHNAME EINGEBEN"
40 INPUT NN$
50 PRINT "GUTEN TAG";
60 PRINT VN$;
70 PRINT " ";
80 PRINT NN$;
```

Vielleicht fragen Sie sich, was die seltsamen Variablennamen hier sollen. Nun, wir wollen Sie langsam an ungefährliche und eindeutige kurze, d.h. maximal zweistellige Variablennamen gewöhnen. In diesem Falle bietet es sich an, denn VORNAME geht ja nicht, und so kürzen wir beide eben entsprechend den jeweiligen ersten Buchstaben der Hauptbestandteile der Wörter ab. Übrigens dachten wir einen Augenblick daran, NAME1$ und NAME2$ zu verwenden. Aber dann fiel uns unser eigener Hinweis ein... Die Zahlen können noch so unterschiedlich sein. An der fünften Stelle von NAME1$ und NAME2$ helfen sie gar nicht mehr. Denn beide Variablennamen beginnen mit NA und sind daher für den C64 völlig identisch.

Bleiben wir also bei VN$ und NN$!

Das Semikolon im PRINT-Befehl: Alles schön in einer Reihe!

Dieses Beispiel lehrt uns nicht nur die Vorteile von Stringvariablen, sondern auch, daß ein Semikolon am Ende einer Zeile große Wirkung haben kann. Es bewirkt hinter einem PRINT-Befehl, daß BASIC keine neue Zeile beginnt. Sonst sähe die Anrede ja auch nicht besonders schön aus.

Können Sie noch einen echten Hammer ertragen? Wenn nein, so überspringen Sie bitte den Rest des Absatzes. -

Na?

Fein, daß Sie weiter mitmachen. Was halten Sie denn von folgendem Programm?

```
10 PRINT "Bitte Vorname eingeben"
20 INPUT VN$
30 PRINT "Bitte Nachname eingeben"
40 INPUT NN$
50 GN$ = VN$ + " " + NN$
60 PRINT "Guten Tag ";
70 PRINT GN$
```

Hierbei bedeutet GN Gesamtname!

Es darf nicht funktionieren. Es kann nicht funktionieren, und es funktioniert doch. Bei der Addition von VN$ und NN$ hat BASIC anscheinend keine Schwierigkeiten. Und was sollen die beiden Anführungszeichen mit dem Leerzeichen dazwischen? Auch ein " " ist ein String, also eine Zeichenkette, die allerdings nur aus einem Leerzeichen besteht. Trotzdem kann sie problemlos einer Stringvariablen zugewiesen werden. Bei der Addition von Stringvariablen hängt BASIC diese einfach hintereinander.

Um Ihnen in einem letzten Programm zu den Stringvariablen zu beweisen, daß diese praktisch alle Zeichen aufnehmen können

(bis auf das Anführungszeichen, das diese ja einschließen muß), haben wir uns folgendes Programm ausgedacht:

```
10 INPUT ZEICHEN$
20 ZEICHEN$ = ZEICHEN$ + ZEICHEN$ + ZEICHEN$
30 ZEICHEN$ = ZEICHEN$ + ZEICHEN$ + ZEICHEN$
40 ZEICHEN$ = ZEICHEN$ + ZEICHEN$ + ZEICHEN$
50 ZEICHEN$ = ZEICHEN$ + ZEICHEN$ + ZEICHEN$
60 PRINT ZEICHEN$
```

Geben Sie nach dem Start einfach ein beliebiges Zeichen ein, und Sie erhalten ein ganze Menge davon. Das liegt daran, daß BASIC in Zeile 20 aus dem einem Zeichen drei Zeichen macht, aus diesen drei in Zeile 30 neun Zeichen und so weiter. Mehr als drei Zeichen sollten Sie aber hinter dem Fragezeichen nicht eingeben, weil BASIC Ihnen sonst einen Fehler meldet. Klar, alles hat seine Grenzen.

3.3 Wie man sich lästige Tipperei erspart: Das Kopieren und Löschen von Programmzeilen

Vielleicht haben Sie sich beim Abtippen des letzten kleinen Beispielprogramms über die langen Zeilen, die sich nur durch die Zeilennummer unterscheiden, geärgert. Wir haben übrigens beim Schreiben dieser Zeilen viel weniger Arbeit gehabt und eigentlich nur 3 Zeilen getippt. Das funktioniert folgendermaßen: Nach dem Eintippen der Zeile 20 und Drücken der Return-Taste sind wir mit dem Cursor erneut auf diese Zeile gegangen, haben die 20 in 30 geändert und wieder die Return-Taste gedrückt. Anschließend haben wir das noch mit 40 und 50 gemacht, und fertig waren drei Zeilen. Identische Zeilen kann man also ganz einfach dadurch produzieren, daß man die Zeilennummer ändert.

Genauso einfach kann man Zeilen löschen, indem man einfach die Zeilennummer hinschreibt und die Return-Taste drückt. Dadurch ersetzt man die bisherige Zeile mit dieser Nummer also durch eine leere Zeile, die von BASIC einfach unterschlagen wird.

Alles klar? Fein, dann können wir ja die neuen Erkenntnisse abschließend zusammenfassen:

3.4 Zusammenfassung

1. Mit Variablen (wörtlich: Veränderliche) sind zusammenhängende Zeichengruppen gemeint, die als Platzhalter für bestimmte Größen in einem Programm verstanden werden können. In dem Beispiel VERBRAUCH = LITER / KILOMETER * 100, das wir hier als Programmzeile verstehen wollen, stehen die drei Begriffe VERBRAUCH, LITER und KILOMETER für bestimmte Werte, die entweder eingegeben oder berechnet werden. Diese Werte für den Benzinverbrauch, die getankten Liter und die gefahrenen Kilometer sind grundsätzlich variabel, d.h. beliebig im Rahmen einer konkreten Berechnung veränderlich. Verbunden werden sie durch eine Formel. Aus dem Beispiel ergibt sich bereits, daß solche Variablen die wichtigsten Grundlagen eines jeden Programms sind. Man könnte sie auch als Schubladen verstehen, in denen jeweils ein bestimmter Wert, der sich wieder per Berechnung aus anderen Variablen ergeben kann, abgelegt wird.

2. Das Beispiel AUSGABEN = AUSGABEN + 20 zeigt, daß das Gleichheitszeichen in einer Formel mit Variablen nicht Identität ausdrückt, sondern eine Zuweisung. In diesem Fall wird die Schublade mit dem Namen AUSGABEN mit dem Wert gefüllt, der vorher darin war, vermehrt um den Wert 20.

   Um das ganz klar im Programm stehen zu haben, kann man auch den LET-Befehl verwenden:

   ```
   LET AUSGABEN = AUSGABEN + 20
   ```

3. Man unterscheidet grundsätzlich numerische und alphanumerische Variablen oder Stringvariablen. Numerische Variablen können nur Zahlenwerte aufnehmen. Dafür kann man mit ihnen die üblichen Berechnungen durchführen.

4. Stringvariablen stehen für Namen und Begriffe. Sie werden durch ein $ am Schluß gekennzeichnet und können bis auf das Anführungszeichen beliebige Zeichen und Zeichenketten enthalten. Diese müssen bei der Zuweisung in Anführungsstriche gesetzt werden (VN$ = "PETER"). Der Befehl (direkt oder im Programm)

   PRINT VN$ bringt dann den gewünschten Begriff (hier den Namen PETER) auf den Bildschirm.

5. Mit Stringvariablen kann man nicht rechnen. Das +-Zeichen zwischen zwei Strings bewirkt, daß BASIC diese einfach aneinanderhängt.

6. Für die Wahl von Variablennamen gilt folgendes:

   Sie dürfen theoretisch über zwei Bildschirmzeilen gehen, aber kein Leerzeichen enthalten.

   Sie sollten nur Buchstaben (a - z) und Ziffern (0 - 9) enthalten.

   Sie dürfen nicht mit einer Zahl beginnen und auch keine BASIC-Wörter enthalten.

7. Programmzeilen kann man einfach dadurch löschen, daß man ihre Nummer auf den Bildschirm schreibt und anschließend direkt die Return-Taste betätigt.

8. Leicht verdoppelt bzw. beliebig kopiert werden können Programmzeilen, indem man ihre Zeilennummer mit der gewünschten neuen Nummer überschreibt und die Return-Taste betätigt.

9. Mit Hilfe des Semikolons in einem PRINT-Befehl können Sie dafür sorgen, daß mehrere Angaben in einer Zeile erscheinen:

   Die Zeile:

```
10 PRINT LITER;KILOMETER;VERBRAUCH
```

   gibt den Inhalt der jeweiligen Variablen, ihren Wert also, nacheinander in einer Zeile aus.

3.5 Übungen zur Sicherung des Erreichten

Sie haben nun schon einen ersten Einblick in die Programmierung mit BASIC gewonnen. Und auch einige kleine nützliche Anwendungen haben wir dabei eingestreut. In diesem Kapitel wollen wir noch mehr Beispiele zusammenstellen, die Ihnen erstens zeigen sollen, wie nützlich BASIC ist, und Ihnen zweitens Anregungen geben, selbst noch mehr auszuprobieren. Wenn Sie so neben nützlichen Programmen noch ein wenig sicherer im Umgang mit BASIC werden, ist das ein zusätzlicher Erfolg.

3.5.1 Mehrwertsteuer

Stellen Sie sich vor, Sie haben die Möglichkeit, in einem Großmarkt einiges einzukaufen, was Sie für Ihren C64 gut gebrauchen können, etwa Computerpapier und Disketten. Das Problem ist nur, daß die Preise, die Sie vorfinden, die Mehrwertsteuer noch nicht enthalten. Diese muß noch dazugerechnet werden.

Das folgende Programm soll Ihnen zu jedem Preis die Mehrwertsteuer und den neuen Gesamtpreis anzeigen. Da auch die Mehrwertsteuer variabel sein soll, benötigen wir insgesamt drei Variablen: Preis, Mehrwertsteuer und Gesamtpreis. Mit dieser Vorüberlegung ist unser Programm auch schon fast geschrieben:

```
10 PRINT "CLR"
20 PRINT "MEHRWERTSTEUER"
30 MEHRWERTSTEUER = .14
40 PRINT "BITTE PREIS EINGEBEN"
50 INPUT PREIS
60 GESAMTPREIS = PREIS + MEHRWERTSTEUER * PREIS
70 PRINT "DDER GESAMTPREIS IST:";
80 PRINT GESAMTPREIS
```

Nun zur Erklärung unseres Programms:

In Zeile 10 wird der Bildschirm gelöscht. Anschließend wird in Zeile 20 das Stichwort zu diesem Programm auf dem Bildschirm ausgegeben. Das werden Sie in allen folgenden Programmen wiederfinden, da man als Benutzer sofort prüfen kann, ob man auch

das richtige Programm geladen hat. In Zeile 30 wird die aktuelle Mehrwertsteuer festgelegt. Dadurch ist unser Programm bei diesbezüglichen Änderungen flexibel.

In Zeile 40 und 50 erfolgt dann die eigentliche Eingabe. Sie wissen ja noch, daß ein Hinweis auf dem Bildschirm die Deutung des Fragezeichens (das beim INPUT erscheint) ungemein erleichtert. Nun enthält also die Variable PREIS den Betrag, der umgerechnet werden soll. Diese Rechnung findet dann in Zeile 60 statt, die schon etwas komplizierter ist. Die Mehrwertsteuer wird ja durch Multiplikation des Betrages mit dem Steuersatz (hier 14% = 14 Hunderstel = 0.14) berechnet. Daher steht in unserer Programmzeile auch "Mehrwertsteuer * Preis". Diese Mehrwertsteuer muß zum vorherigen Preis addiert werden. (Sie sehen schon: Auch der Computer merkt, daß Preise durch Steuern nur erhöht werden.) Dadurch entsteht dann der neue Gesamtpreis, der der gleichnamigen Variablen zugewiesen wird. Übrigens: Ihr C64 weiß genausoviel wie Ihr ehemaliger Mathematiklehrer - nämlich daß Punktrechnung vor Strichrechnung geht. Deshalb berechnet er erst Mehrwertsteuer * Preis und addiert das Ergebnis anschließend zum Preis. Wenn Sie sich nicht sicher sind, können Sie das, was zuerst gerechnet werden soll, wie gewohnt in Klammern eingeben. Die Zeile würde dann folgendermaßen lauten:

```
60 GESAMTPREIS = PREIS + (MEHRWERTSTEUER * PREIS)
```

Zum Schluß wird in Zeile 70 und 80 nur noch das Ergebnis ausgegeben. Das Semikolon am Ende der Zeile 70 sorgt dafür, daß die folgende Bildschirmausgabe durch PRINT in derselben Zeile erfolgt. Das sieht einfach besser aus.

3.5.2 Rabatt

Beim folgenden Fall gehen wir von umgekehrten Verhältnissen aus. Sie haben einen Händler gefunden, der Ihnen, aus welchen Gründen auch immer, Rabatt gewährt. Das heißt, Sie bekommen einen Nachlaß. In diesem Falle können Sie das Programm ganz leicht entsprechend ändern. Eigentlich brauchen Sie nur den Rabatt in Zeile 30 einzutragen und in Zeile 60 das "+" durch ein "-"

zu ersetzen. Allerdings wirkt es besser, wenn Sie auch die Bildschirmausgaben und Variablen entsprechend ändern. Das Programm könnte dann folgendermaßen aussehen:

```
10 PRINT "CLR"
20 PRINT "RABATTBERECHNUNG"
30 RABATT = .20
40 PRINT "BITTE PREIS EINGEBEN"
50 INPUT PREIS
60 NEUERPREIS = PREIS - RABATT * PREIS
70 PRINT "DER NEUE PREIS IST :";
80 PRINT NEUERPREIS
```

Sie sehen, Steuern bezahlen ist genau so einfach wie Rabatt bekommen. Was uns dabei lieber ist, lassen wir an dieser Stelle besser offen. Sollten Sie übrigens im Geschäft mit besagtem Rabatt eine Ware für 1500.- DM kaufen, so können Sie sich gratulieren. Sie haben dann einen Betrag von 300.- DM eingespart, und es kann gut sein, daß Sie für diesen Preis bereits Ihren C64 (zumindest den Rechnerkasten selbst) bekommen haben.

Wenn Sie das Ausmaß Ihres nächsten Schnäppchens wieder mit BASIC berechnen wollen, so speichern Sie dieses Programm z.B. unter dem Namen RABATTBERECHNUNG auf Ihrer Arbeitsdiskette ab. Wir werden bei Programmen, die wir in diesem Buch nicht mehr wieder aufnehmen und weiter entwickeln, nicht immer eigens darauf verweisen. Nur hinweisen möchten wir Sie noch darauf, daß es ganz günstig wäre, die Zeile 30 umzuändern in

```
INPUT RABATT
```

und vorher eine erläuternde Zeile einzufügen:

```
25 PRINT"BITTE RABATT EINGEBEN, Z.B.:0.20"
```

Dann können Sie zum einen flexibel auf verschiedenste Angebote reagieren, und zum anderen wissen Sie, was der C64 an dieser Stelle von Ihnen wissen will.

3.5.3 Die Gehaltserhöhung

In der letzten Woche haben Sie zufällig Ihrem Chef erzählt, daß Sie sich einen C64 gekauft haben und seit kurzem eigene Programme in BASIC schreiben. Ihr Chef war sofort begeistert und hat Ihnen eine Gehaltserhöhung angeboten. Die Summe war beträchtlich, aber Sie erinnern sich noch an die letze Erhöhung, bei der nach Abzug aller Abgaben nicht mehr viel übrigblieb. Deshalb wollen Sie natürlich schnell einmal ausrechnen, wieviel Ihnen denn netto übrigbleibt.

Für die Berechnung nehmen Sie natürlich wieder BASIC. Das Programm ähnelt den beiden letzten Programmen, nur wird diesmal nicht ein Anteil (Rabatt) abgezogen, sondern mehrere.

Wir gehen davon aus, daß vom Bruttogehalt folgende Abzüge gemacht werden:

- Steuer
- Krankenversicherung
- Rentenversicherung
- Arbeitslosenversicherung
- Kirchensteuer

Diesmal wollen wir uns an Abkürzungen gewöhnen, denn schon mit dem Variablennamen STEUER haben wir wegen des Bestandteils ST kaum behebbare Schwierigkeiten.

Und zum anderen sind einige Variablennamen sonst doch etwas lang, um sie mehrmals tippen zu müssen. Außerdem gewöhnen Sie sich so langsam an eine Programmierweise, bei der man mit kurzen Variablennamen auskommt.

Folgende Abkürzungen sollen verwendet werden:

BR steht für BRUTTO.
KV steht für KRANKENVERSICHERUNG.
AV steht für ARBEITSLOSENVERSICHERUNG.
LS steht für LOHNSTEUER.
KS steht für KIRCHENSTEUER.
RV steht für RENTENVERSICHERUNG.
NE steht für NETTO.

Das Programm könnte dann also folgendermaßen aussehen:

```
10 PRINT "CLR"
20 INPUT "BRUTTOBETRAG"; BR
30 LS = 20/100*BR
40 KV = 6.5/100*BR
50 RV = 4.5/100*BR
60 AV = 2/100*BR
70 KS = LS/10
80 NE= BR - LS - KV - RV
90 NE= NE- AV - KS
100 PRINT "NETTO="; NE
```

Erklärung des Programms

In Zeile 10 wird wie üblich der Bildschirm gelöscht. Anschließend wird der Bruttoverdienst abgefragt und in der Variablen BR gespeichert. Nun werden die einzelnen Abzüge berechnet. Wir haben sie mit Absicht in der Form "20/100*BR" aufgeschrieben, weil die 20 dann genau die Prozente darstellt. Dadurch ist es leichter, diesen Wert zu ändern, als wenn dort Werte wie "0.2*BR" oder "0.065*BR" stünden. In der Zeile 70 haben wir die Kirchensteuer KS nicht durch Multiplikation mit BR, sondern als 10% von der STEUER (LS) berechnet. Somit ändert sich die Kirchensteuer automatisch, wenn wir einen anderen Wert für die Steuer einsetzen.

In den Zeilen 80 und 90 werden dann die Beträge abgezogen, und damit wird das Nettogehalt (NE) berechnet. Damit Sie die Richtigkeit des Programms gleich prüfen können, wollen wir ein Beispiel durchrechnen lassen.

Angenommen, Sie verdienen im Augenblick DM 3000.-. Dann berechnet das Programm einen Nettoverdienst von DM 1950.-. Wenn Ihnen Ihr Chef nun für die Einarbeitung in BASIC eine Gehaltserhöhung von DM 300.- anbietet, verdienen Sie anschließend DM 2145.-. Wenn Sie nun direkt wissen wollen, wieviel Ihnen von den DM 300.- mehr denn übriggeblieben ist, so geben Sie einfach im Direktmodus (also ohne Zeilennummer) ein:

```
PRINT 2145 - 1950 (plus Return-Taste)
```

Das Ergebnis lautet DM 195.-, und wir können Ihnen nur wünschen, daß Ihnen bei den vielen Abgaben wirklich soviel von einer Gehaltserhöhung übrigbleibt.

4. Was man für größere Programme noch braucht: Weitere wichtige BASIC-Befehle

Wir haben nun schon kleine BASIC-Programme gemeinsam entwickelt, aber uns fehlen noch einige Befehle, um bessere und größere Programme zu schreiben. Diese neuen Befehle wollen wir Ihnen in diesem Kapitel näherbringen.

4.1 BASIC kann springen: GOTO

Wir nehmen einmal an, Sie wollen im Sommer in die Vereinigten Staaten fahren. Sie gehen zur Bank, tauschen Ihr Geld in Dollar um, aber erhalten leider keins dieser praktischen Umrechnungskärtchen. Dafür sagt Ihnen die Bank den Umrechnungskurs: 1 Dollar sind 1,89 DM. Nun können Sie natürlich zu Hause schnell ein Programm schreiben, das Ihnen die lästige Umrechnung abnimmt. Dieses könnte etwa folgendermaßen aussehen:

```
10 PRINT "CLR"
```

Zur Erinnerung: Mit CLR ist hier die Taste gemeint, auf der CLR und Home steht. Wenn Sie sie zusammen mit der Shift-Taste drücken, bekommen Sie eine Art schwarzes Herz auf weißem Grunde!

```
20 INPUT "Dollar "; DOLLAR
30 MARK = DOLLAR * 1.89
40 PRINT MARK
```

Wenn Sie das Programm mit RUN starten, wird Ihnen jeweils ein Dollar-Wert in DM umgerechnet. Wenn Sie nun aber viele Werte berechnen wollen, ist es lästig, nach jedem Programmende stets RUN eingeben zu müssen. Fügen Sie deshalb einfach eine Zeile

```
50 GOTO 20
```

ein. Wenn Sie das so geänderte Programm probeweise starten, fragt BASIC nach dem ersten Rechendurchlauf sofort nach ei-

nem neuen DOLLAR-Wert. Das geht solange weiter, bis Sie das Programm in der folgenden Weise abbrechen: Sie halten bitte die Taste ganz links gedrückt, auf der die Befehle RUN und STOP stehen. Dann drücken Sie zusätzlich die RESTORE-Taste. Sie befindet sich über der Return-Taste. Sofort erscheint oben links auf dem Bildschirm ein READY, und Sie haben das Programm verlassen. Der Unterschied besteht darin, daß Ihr Programm nicht bei Zeile 40 endet, sondern BASIC in Zeile 50 durch das GOTO den Befehl erhält, bei der Zeilennummer 20 weiterzumachen. In der Computersprache sagt man auch gerne: In Zeile 50 springt das Programm zu Zeile 20.

4.2 BASIC erfüllt Bedingungen: IF ... THEN

Wir werden später noch ausführlicher auf die Möglichkeiten eingehen, die Ihnen der Befehl GOTO bietet. Im Augenblick stört es Sie aber sicher, daß Sie das Programm umständlich mit RUN/STOP und RESTORE beenden müssen. Es wäre doch viel angenehmer, wenn Sie das erneute Ausführen des Programms von einer Bedingung abhängig machen könnten. Dann könnte man nämlich nach jedem umgerechneten Dollarwert fragen, ob ein weiterer Durchlauf des Programms erwünscht ist. Nur in diesem Fall dürfte der Befehl GOTO 20 ausgeführt werden.

Zu diesem Zweck gibt es den Befehl IF THEN, der eigentlich aus zwei Befehlswörtern besteht. Das erste Wort IF prüft eine Bedingung, und das zweite Wort THEN führt bei "bestandener Prüfung" einen Befehl aus. Wir wollen unser Programm nun einmal um diese Abfrage erweitern:

```
10 PRINT "CLR"
20 INPUT "Dollar "; DOLLAR
30 MARK = DOLLAR * 1.89
40 PRINT MARK
50 INPUT "MOECHTEN SIE NOCH EINEN WERT BERECHNEN (J/N) "; EINGABE$
60 IF EINGABE$ = "J" THEN GOTO 20
```

In Zeile 50 verlangt das Programm eine Eingabe, die in der Variablen EINGABE$ gespeichert wird. Beachten Sie bitte, daß J und N keine Zahlen, sondern Zeichen sind, so daß wir zur Ab-

lage dieser Zeichen eine Stringvariable nehmen müssen ($-Zeichen am Ende des Variablennamens). Nun können wir in Zeile 60 den neuen Befehl zur Prüfung einer Bedingung einsetzen. Wir prüfen, ob die Eingabe ein J war (IF EINGABE$ = "J") und machen den Sprung zur Zeile 20 davon abhängig (THEN GOTO 20).

Probieren Sie Ihr Programm jetzt ruhig noch einmal aus, indem Sie verschiedene Dollarwerte bequem abfragen. Immer werden Sie zunächst nach einem Dollarwert gefragt. Der entsprechende Mark-Wert wird Ihnen dann angezeigt. Gleichzeitig erscheint die gewünschte Abfrage, die Sie mit J beantworten müssen. Dann geht es weiter. Das ist schon sehr viel bequemer, als es jeweils mit RUN neu starten zu müssen.

Was aber macht das Programm, wenn ein N eingegeben wurde. Probieren Sie auch das ruhig aus. Sofort erscheint ein READY. Halt, werden Sie sagen, für diesen Fall haben wir doch gar keinen Befehl erteilt. Nun hier gilt folgendes: Ist die mit IF abgefragte Bedingung nicht erfüllt, so beendet BASIC die aktuelle Zeile und macht in der nächsten Zeile weiter. Da diese aber nicht vorhanden ist, wird unser Programm beendet.

Wie bekommt man eine ganze Tabelle von Daten?

Nun haben wir unser Programm schon um einige Möglichkeiten erweitert und angenehmer gestaltet. Wir wollen aber die Berechnung für eine ganze Reihe von Werten durchführen lassen und nicht jeden Dollarwert neu eingeben müssen. Das ist mit unserem bisherigen Wissen leicht zu bewerkstelligen. Wir starten einfach mit einem Dollarwert, lassen die DM berechnen, erhöhen DOLLAR um 1 und lassen das Programm erneut rechnen. Dabei wollen wir nur verhindern, daß BASIC bis zum nächsten Stromausfall rechnet, und fragen daher mit IF ab, ob der Wert für DOLLAR eine bestimmte Grenze überschreitet. Das zugehörige Programm sieht dann folgendermaßen aus:

```
10 PRINT "CLR"
20 DOLLAR = 1
30 MARK = DOLLAR * 1.89
40 PRINT DOLLAR,MARK
50 DOLLAR = DOLLAR + 1
60 IF DOLLAR < 20 THEN GOTO 30
```

Was haben wir hier gemacht?

Statt der Abfrage durch INPUT erhöhen wir in Zeile 50 den bisherigen Wert für DOLLAR um 1. Da wir zu Beginn (Zeile 20) mit 1 starten, wird er also beim ersten Mal in Zeile 50 auf 2, beim nächsten mal auf 3 und so fort erhöht. In Zeile 60 prüfen wir nun, ob DOLLAR noch immer kleiner als 20 ist. Ist dies der Fall, so springt das Programm zur Zeile 30 und berechnet erneut die DM. Hat DOLLAR aber den Wert 20 angenommen, endet das Programm.

Wichtig ist, daß wir uns in Zeile 40 zunächst die DOLLAR und dann erst die umgerechneten Markbeträge ausgeben lassen. Denn sonst bekämen wir ja keine schöne Tabelle.

Drucken: Umlenken der Daten vom Bildschirm auf den Drucker

Wenn Sie das Programm starten, haben Sie anschließend eine erste Tabelle für die Umrechnung auf dem Bildschirm. Sie können sich nun die Daten ausdrucken lassen und anschließend in die Vereinigten Staaten fahren.

Zu diesem Zweck brauchen Sie nur die folgenden Zeilen an den Anfang des Programms zu schreiben:

```
5 OPEN1,4
6 CMD1
```

Diese beiden Befehle müssen Sie am Ende des Programms wieder aufheben. Das geschieht durch die beiden Zeilen:

```
65 PRINT#1
66 CLOSE1
```

Wenn Sie jetzt RUN (Return) eingeben, werden die Daten nicht auf den Bildschirm, sondern auf den Drucker ausgegeben, und Sie können sie im Gegensatz zu Ihrem Computersystem schön und bequem mitnehmen.

Übrigens können Sie leicht auch Ihre Mitfahrer mit Umrechnungslisten versorgen, indem Sie das Programm einfach so oft starten, wie Sie Ausdrucke benötigen. Na, wenn das nichts ist!!!

Am besten speichern Sie diese Version unter einem anderen Namen ab, so daß Sie bei Bedarf auf beide zurückgreifen können. Später werden Sie Möglichkeiten kennenlernen, wie man beide Möglichkeiten in einem Programm gleichzeitig zur Verfügung halten kann.

Möglichkeiten beim Vergleich von Daten: Vergleichsmaßstäbe

Wir wollen Ihnen aber noch einen weiteren äußerst nützlichen Befehl vorstellen, der das eben erstellte Programm vereinfacht.

Vorher aber noch ein kleiner Nachtrag zu IF THEN:

Es gibt neben dem Gleichheitszeichen (=) und dem Kleiner-als-Zeichen (<) noch weitere Zeichen und Zeichenkombinationen, die gewissermaßen Prüfnormen angeben. Man nennt sie logische Operatoren, weil sie etwas bewirken, dabei aber logisches Denken verlangen (gestehen wir Ihrem mit BASIC arbeitenden C64 hier durchaus die Fähigkeit zum logischen Denken zu!).

Die einfachen Vergleichsmaßstäbe sind:

= `IF VERBRAUCH = 10 THEN PRINT "Alles OK! Normalverbrauch"`

< `IF VERBRAUCH < 10 THEN PRINT "Hey toll! Benzin gespart"`

> `IF VERBRAUCH > 10 THEN PRINT "Au weih! Werkstatt nötig!"`

\>= Das bedeutet: "größer oder gleich"

<= Das ist das Gegenteil: "kleiner oder gleich"

<> Dies Zeichen steht für ungleich!

Man kann Vergleichsmaßstäbe auch verknüpfen:

```
IF VERBRAUCH > 10 AND FAHRSTRECKE$ = "STADTVERKEHR" THEN PRINT
"Benzinverbrauch zu hoch, aber Stadtverkehr!"

IF VERBRAUCH > 10 OR KILOMETER > 500 THEN PRINT
"Wagen überprüfen!"
```

Nun aber zu dem angekündigten weiteren praktischen BASIC-Befehl.

4.3 BASIC läßt sich Grenzen setzen: FOR ... NEXT

In unserem Programm haben wir so etwas wie eine Schleife erstellt. Ein Teil des Programms wurde nämlich viele Male (19) abgearbeitet. Für solche Fälle stellt uns BASIC einen speziellen Befehl zur Verfügung, der uns Schreibarbeit erspart und das Programm übersichtlicher macht. Statt

```
20 DOLLAR = 1
50 DOLLAR = DOLLAR + 1
60 IF DOLLAR < 20 THEN GOTO 30
```

können wir einfach FOR DOLLAR = 1 to 20 und NEXT DOLLAR schreiben. Damit sieht unser Programm dann folgendermaßen aus:

```
10 PRINT "CLR"
20 FOR DOLLAR = 1 TO 20
30 MARK = DOLLAR * 1.89
40 PRINT DOLLAR,MARK
50 NEXT DOLLAR
```

Für BASIC besagt die Zeile FOR DOLLAR = 1 TO 20 folgendes: Nimm dir beim ersten Mal die Variable DOLLAR und setze sie auf 1. Dann führe alle Zeilen aus, bis du auf ein NEXT DOLLAR triffst. Erhöhe die Variable DOLLAR um 1 und "springe" wieder in die Zeile, wo das FOR stand. Prüfe, ob DOLLAR noch nicht den Wert 20 überschritten hat (TO 20). Ist das der Fall, so suche die Zeile mit dem NEXT DOLLAR und mache in der darauffolgenden Zeile weiter, ansonsten arbeite wieder die Zeilen bis zum NEXT DOLLAR ab (Das, was zwi-

schen FOR und NEXT an Programmzeilen steht, nennt man eine Schleife).

Wenn Ihnen der Einsatz solcher FOR-NEXT-Schleifen noch nicht ganz klar ist, brauchen Sie den Absatz nicht so lange zu lesen, bis Sie ihn verstanden haben. Wir werden im folgenden noch viele Beispiele durchprobieren, und dabei werden Sie schnell ein Gefühl für die neu gelernten Befehle bekommen.

4.4 Zusammenfassung

In diesem Kapitel haben wir bis jetzt die folgenden drei Befehle kennengelernt, die man für etwas größere Programme unbedingt braucht:

1. Der Befehl GOTO 20 bedeutet, daß BASIC, sobald es zu diesem Befehl gelangt, nicht Zeile für Zeile weiterläuft, sondern zur Programmzeile 20 springt und dort weitermacht. Diese Programmzeile muß natürlich auch vorhanden sein, sonst gibt es eine Fehlermeldung.

2. Der Befehl IF THEN besteht eigentlich aus zwei Befehlswörtern. Der erste Bestandteil (IF) prüft eine Bedingung, der zweite (THEN) nennt einen Befehl, der "bei bestandener Prüfung" ausgeführt werden soll. Wichtig sind zum Beispiel Abfragen wie INPUT EINGABE$ und die anschließende Bedingungszeile:

   ```
   IF EINGABE$ = "J" THEN GOTO 20.
   ```

   Wichtig in diesem Zusammenhang sind die folgenden Vergleichsoperatoren, die dem Vergleich von Daten dienen:

   das Gleichzeitszeichen: IF A = 20 THEN GOTO 100

   das Größer-als-Zeichen: IF A > 20 THEN GOTO 100

   das Kleiner-als-Zeichen: IF A < 20 THEN GOTO 100

   Möglich sind auch Kombinationen:

   Kleiner oder gleich: IF A <= 20 THEN GOTO 100

   Größer oder gleich: IF A >= 20 THEN GOTO 100

ungleich: IF A <> 20 THEN GOTO 100

Auch Doppelabfragen sind möglich:

```
IF A > 20 AND B < 20 THEN ...
```

In diesem Falle müssen beide Bedingungen erfüllt sein.

Oder:

```
IF A > 20 OR B > 20 THEN ...
```

In diesem Falle reicht es, wenn eine der beiden Bedingungen erfüllt ist.

3. Der Doppelbefehl FOR NEXT führt zu einer sogenannten Programmschleife. In der Startzeile der Schleife steht zum Beispiel FOR I = 1 to 10. In der Zielzeile der Schleife steht dann NEXT I. Das, was zwischen diesen beiden Zeilen steht, wird dann zehnmal abgearbeitet, wobei jedesmal I um eins erhöht wird. Das heißt: Der Computer zählt seine Schleifendurchgänge mit, und sobald er die gesetzte Grenze erreicht hat, verläßt er den Schleifenbereich und macht dahinter weiter.

4. Drucken können Sie Daten innerhalb eines Programms, indem Sie vier Zeilen ins Programm aufnehmen, die die Daten vom Bildschirm auf den Drucker umlenken. Vor dem PRINT-Befehl müssen die folgenden beiden Zeilen im Programm untergebracht werden:

```
OPEN1,4
CMD1
```

Und nach dem PRINT-Befehl müssen diese beiden Befehle im Programm rückgängig gemacht werden. Das geschieht mit:

```
PRINT#1
```

und

```
CLOSE1
```

5. Es geht nichts über die Praxis! Anwendung der neuen Befehle

Nachdem wir die letzten wichtigen Grundbefehle des Programmierens in BASIC kennengelernt haben, sollten wir sie im folgenden praktisch umsetzen. Dabei werden wir noch einmal auf unser bekanntes Beispiel zurückgreifen, Ihnen dann aber noch einiges mehr vorstellen. Sie kommen dann sicherlich selbst bald auf den Geschmack, was Sie mit den Befehlen, die Sie kennengelernt haben, anfangen können. Die wichtigsten haben Sie dann im Griff. Und der Rest kommt durch Phantasie, Ausprobieren und zunehmende Erfahrung. Anders ist es bei uns auch nicht gelaufen.

5.1 Erste Anwendung der neuen Befehle: Die Benzinkostentabelle

Wir haben ja schon gemeinsam ein Programm geschrieben, das aus gefahrenen Kilometern und getankten Litern den Benzinverbrauch berechnet. Da Sie aber in Urlaub fahren wollen und Ihren C64 nicht mitnehmen, wäre eine Tabelle nützlich, in der Sie sofort den Verbrauch nachschauen können. Eine solche Tabelle können wir mit den bisher gelernten Befehlen problemlos erstellen. Allerdings taucht dabei ein Problem auf: Wir haben zwei veränderliche Werte (Liter und Kilometer), aber in einer einfachen Tabelle kann sich immer nur ein Wert ändern. Später werden wir sehen, wie man dieses Problem geschickt lösen kann, aber im jetzigen Beispiel wollen wir einfach von jeweils 300 Kilometern ausgehen und damit voraussetzen, daß Sie stets nach 300 Kilometern tanken. (Hoffentlich ist da immer eine Tankstelle! Aber Sie werden ja bald in BASIC weiter sein, und dann können Sie nach Belieben tanken.)

Nun müssen wir noch kurz überlegen, welche Werte die Tabelle enthalten soll. Angenommen, Ihr Wagen verbraucht mindestens 6 Liter auf 100 Kilometer und höchstens 10. Dann sollte die Tabelle also von 18 bis 30 Liter gehen. Damit lautet der Anfang unserer Schleife schon einmal: FOR LITER = 18 TO 30. Inner-

halb der Schleife wollen wir dann den VERBRAUCH berechnen und anschließend LITER und VERBRAUCH auf dem Bildschirm ausgeben. Danach können wir die Schleife mit: NEXT LITER abschließen. Das Programm für die gewünschte Tabelle sieht dann also so aus:

```
10 PRINT "CLR"
20 FOR LITER = 18 TO 30
30 VERBRAUCH = LITER / 300 * 100
40 PRINT LITER,VERBRAUCH
50 NEXT LITER
```

Übrigens: Achten Sie auf das Komma, das hier in Zeile 40 auftaucht. Es produziert eine schöne Tabelle, bei der links die Liter und rechts der damit verbundene Verbrauch aufgelistet werden.

Wenn Sie das Programm mit SAVE "BENZTAB",8 auf Diskette gespeichert haben und starten, erhalten Sie folgende Ausgabe auf dem Bildschirm:

```
18          6
19          6.33333333
20          6.66666667
21          7
22          7.33333334
23          7.66666667
24          8
25          8.33333334
26          8.66666667
27          9
28          9.33333333
29          9.66666667
30          10
```

Der Ausdruck der Programmergebnisse

Übrigens haben Sie hier eine gute Möglichkeit zu üben, wie man durch ein paar "Handgriffe" die Ergebnisse ausdrucken läßt. Als Ergebnis dürfte sich das Folgende präsentieren (wenn Sie unseren Vorschlägen zur Umlenkung von Daten vom Bildschirm auf den Drucker gefolgt sind):

```
10 OPEN1,4
15 CMD1
20 FOR LITER = 18 TO 30
30 VERBRAUCH = LITER / 300 * 100
40 PRINT#1,LITER,VERBRAUCH
50 NEXT LITER
55 PRINT#1
60 CLOSE1
```

Ein solches Programm können Sie jetzt unter einem entsprechenden Namen abspeichern, wenn Sie später wieder mal einen Ausdruck brauchen.

Variablen an den Anfang des Programms

Wir wollen uns hier nun dem alten Programm ohne die Druckmöglichkeit wieder zuwenden:

Wenn Sie es in einigen Wochen laden, um es mit anderen Werten zu starten, so werden Sie feststellen, daß das gar nicht einfach ist. Sie müssen die Änderungen nämlich alle mitten im Programm vornehmen. Wollen Sie beispielsweise die KILOMETER von 300 auf 400 erhöhen, so muß das in Zeile 30 erfolgen. Wollen Sie eine Tabelle von 18 bis 36 Liter erstellen, so müssen Sie in der Zeile 20 die FOR-NEXT-Schleife ändern. Viel übersichtlicher wird ein Programm, wenn Sie zum Programmbeginn Variablen definieren und diese statt einfacher Zahlen verwenden. Dann sehen Sie am Variablennamen sofort, was in einer Rechnung geschieht.

Statt einfacher Zahlen Variablen zu benutzen, hat noch einen weiteren Vorteil. Stellen Sie sich vor, Ihr Programm wird irgendwann 3 Seiten lang, und zehnmal wird mit der Anzahl der Kilometer gerechnet. Dann müssen Sie für die Änderung an 10 Stellen im Programm eine Zahl ändern. Wenn statt dessen am Anfang des Programms

```
KILOMETER = 300
```

steht, brauchen Sie nur einmal den Wert auf 400 zu ändern. Wir stellen Ihnen deshalb noch einmal das gleiche Programm vor - allerdings mit den beschriebenen Änderungen.

```
10 PRINT "CLR"
20 KILOMETER = 300
30 BEGINNLITER = 18
40 SCHLUSSLITER= 30

50 FOR LITER = BEGINNLITER TO SCHLUSSLITER
60 VERBRAUCH = LITER / KILOMETER * 100
70 PRINT LITER, VERBRAUCH
80 NEXT LITER
```

Der "Nicht-Befehl" REM: Hier hört der C64 nicht auf Sie!

Wir haben mit Absicht mitten im Programm eine Leerzeile gelassen, die Sie natürlich nicht erzeugen sollen. Vielmehr soll sie Ihnen zeigen, wo die Festlegung (Definition) der Variablen endet und das eigentliche Programm beginnt.

Es gibt übrigens auch eine Möglichkeit, hilfreiche Bemerkungen fest in ein Programm einzubauen.

Mit dem Befehl REM (für englisch: REMARK = Bemerkung) können Sie Ihrem C64 und seinem BASIC mitteilen, daß das, was dahinter steht, nur für Sie eine Hilfe sein soll und für ihn keine Bedeutung und schon gar keinen Befehlscharakter hat. Man könnte also überspitzt von einer Art "Nicht-Befehl" sprechen, was aber natürlich nicht stimmt. REM weist ja den Computer gerade an, das Folgende nicht zur Kenntnis zu nehmen.

In unserem Falle wäre es also sinnvoll, eine Zeile 45 ins Programm einzubauen, die etwa so aussehen könnte:

```
45 REM BEGINN DES EIGENTLICHEN PROGRAMMS
```

Der Befehl STEP: Der C64 kann auch schrittweise

Nun wollen wir Ihnen noch eine interessante Möglichkeit im Zusammenhang mit FOR-NEXT-Schleifen vorstellen. Wir gehen daher davon aus, daß Sie inzwischen genauer über den Verbrauch Ihres Wagens Bescheid wissen. Sie brauchen daher nicht mehr eine Tabelle zwischen 18 und 30 Litern, sondern nur noch die Werte von 24 bis 27 Litern. Nun können Sie natürlich einfach BEGINNLITER und SCHLUSSLITER auf diese Werte setzen, allerdings wollen Sie gleichzeitig auch eine feinere Abstufung haben. LITER soll also nicht bei jedem Schleifendurchlauf um 1, sondern nur noch um 0.2 erhöht werden (den Bundesumweltminister wird es freuen). Das läßt sich durch eine zusätzliche Angabe in der FOR TO-Zeile erreichen. Der Befehl lautet STEP (= Schrittweite). Das geänderte Programm sieht dann folgendermaßen aus:

```
10 PRINT "CLR"
20 KILOMETER = 300
30 BEGINNLITER = 24
40 SCHLUSSLITER= 27
45 REM BEGINN DES EIGENTLICHEN PROGRAMMS
50 FOR LITER = BEGINNLITER TO SCHLUSSLITER STEP .2
60 VERBRAUCH = LITER / KILOMETER * 100
70 PRINT LITER, VERBRAUCH
80 NEXT LITER
```

Daß hinter STEP keine Null auftaucht, sondern einfach ein Punkt, wundert Sie sicher nicht mehr!

Wird in der FOR TO-Zeile kein STEP befohlen, geht BASIC stillschweigend von "+1" aus. Sie können hinter STEP auch negative Zahlen schreiben und dadurch erreichen, daß die Schleife von hohen zu niedrigen Werten läuft. Dann muß natürlich BEGINNLITER größer als SCHLUSSLITER sein. Wenn Sie das Programm starten, erhalten Sie folgende Bildschirmausgabe:

```
24          8
24.2          8.06666667
24.4          8.13333334
24.6          8.2
24.8          8.26666668
25          8.33333334
25.2          8.40000001
25.4          8.46666668
25.6          8.53333334
25.8          8.60000001
26          8.66666668
26.2          8.73333335
26.4          8.80000001
26.6          8.86666668
26.8          8.93333335
```

Wie man aus krummen Zahlen grade macht: Die INT-Funktion

Sollte Sie übrigens die unnötige Genauigkeit rechts stören, wollen wir Ihnen an dieser Stelle eine grundsätzliche Möglichkeit verraten, wie man so etwas beseitigt.

Zu dem Zweck multiplizieren Sie den entsprechenden Wert zunächst mit Hundert, wenn Sie hinterher eine Genauigkeit auf Hundertstel haben wollen. Anschließend verwenden Sie die sogenannte INT-Funktion. Die macht aus beliebigen Zahlen ganze (Integer-) Zahlen. Anschließend brauchen Sie nur noch die Werte wieder durch 100 zu dividieren, dann haben Sie das gewünschte Ergebnis, bei dem alles weggeschnitten ist, was über die Hundertstel hinausgeht.

In unserem Falle geht es ja um VERBRAUCH.

Also könnten wir die folgenden Zeilen ins Programm einfügen:

```
62 VERBRAUCH=VERBRAUCH*100
64 VERBRAUCH=INT(VERBRAUCH)
66 VERBRAUCH=VERBRAUCH/100
```

Wir packen aber lieber alles in eine einzige Zeile, die dann die Formel enthält:

```
65 VERBRAUCH=INT(100*VERBRAUCH)/100
```

Unser Programm sieht dann so aus:

```
10 PRINT "CLR"
20 KILOMETER = 300
30 BEGINNLITER = 24
40 SCHLUSSLITER= 27
45 REM BEGINN DES EIGENTLICHEN PROGRAMMS
50 FOR LITER = BEGINNLITER TO SCHLUSSLITER STEP .2
60 VERBRAUCH = LITER / KILOMETER * 100
65 VERBRAUCH=INT(100*VERBRAUCH)/100
70 PRINT LITER, VERBRAUCH
80 NEXT LITER
```

Und schon haben wir ein sehr viel schöneres Ergebnis, bei dem rechts die gewünschten Ergebnisse auftauchen.

Wenn Sie es sich jetzt noch mit Hilfe der vorgestellten Möglichkeiten ausdrucken lassen, brauchen Sie nach dem Tanken nur noch auf die ausgedruckte Tabelle zu schauen. Wenn Sie wollen, können Sie auch beispielsweise eine Tabelle von 18 bis 30 Litern mit einer Schrittweite von 0.01 ausdrucken lassen - vorausgesetzt, Sie haben ein paar Farbbänder und einige Packungen Druckerpapier griffbereit.

5.2 Zweite Anwendung der neuen Befehle: Zahlenraten

Nachdem wir uns nun bei den Programmbeispielen mit vielen ernsthaften Problemen beschäftigt haben, wollen wir auch einmal etwas zu unserem Vergnügen programmieren. Vielleicht kennen Sie auch ein Spiel, das uns zum ersten Mal auf einem kleinen, programmierbaren Taschenrechner begegnet ist: Dabei bestimmt der Computer per Zufall eine Zahl, und man muß anschließend versuchen, sie zu erraten. Dabei soll der C64 nur angeben, ob die Zahl größer oder kleiner als unser Rateversuch ist. Haben wir die richtige Zahl eingegeben, soll er noch die Anzahl der Rateversuche angeben.

Die RND-Funktion: Der C64 spielt Schicksal

Um eine Zufallszahl zu erzeugen, gibt es die Funktion RND (X). Dabei ist RND die Abkürzung für das englische Wort "random" (= zufällig). Für X kann man einen beliebigen Wert einsetzen. Diese Funktion erzeugt Zufallszahlen zwischen 0 und 1. Diese kann man mit

```
ZUFALLSZAHL = RND (X)
```

einer Variablen zuweisen. Jede neue Benutzung von RND führt zu einer neuen Zufallszahl zwischen 0 und 1.

Probieren wir es gleich einmal aus.

```
10 ZUFALLSZAHL = RND (1)
20 PRINT ZUFALLSZAHL
```

Nach dem Start mit RUN erhalten wir die Zahl: .542972088!

Sie nicht, Gott sei Dank, dann beherrscht der Zufall nach wie vor die Welt, und alles ist in Ordnung. Wenn Sie sich einmal etwas tiefer in die geheimnisvolle und überraschende Welt des Zufalls einlassen wollen, verbinden Sie doch unseren Programmzweizeiler einfach mit einer kleinen Schleife. Etwa:

```
5 FOR I = 1 TO 10
10 ZUFALLSZAHL = RND (1)
20 PRINT ZUFALLSZAHL
30 NEXT I
```

Und schon haben Sie zehn schöne Zufallszahlen.

Wenn dabei einmal Zahlen der folgenden Art auftauchen sollten:

```
3.90696526E-03
```

Nur keine Panik. Das "-03" am Schluß bedeutet einfach eine entsprechende Zehnterpotenz (darauf weist der Buchstabe E hin): Also 3.90696526 mal 10 hoch minus 3.

Wie drängt man dem C64 ganze Zahlen auf?

Damit erhält die Variable ZUFALLSZAHL allerdings nur Werte zwischen 0 und 1. Da wir aber ungern eine Zahl wie 0.14956 raten wollen, sollen die Zahlen ganze Zahlen zwischen 0 und 999 sein. Dazu nehmen wir die Zufallszahl mal 1000 und lassen die Stellen hinter dem Komma weg. Das erledigt BASIC für uns automatisch, wenn wir eine Fließkommazahl (das sind die Zahlen mit den Kommastellen) einer Integervariablen (ganze Zahl) zuweisen.

Integervariablen erkennt man an dem nachgestellten Prozentzeichen. D.h. werden einer solchen Variablen Werte zugewiesen, werden die Nachkommastellen ignoriert.

Nehmen wir ein Beispiel:

```
10 A=50.34
20 B%=A
30 PRINT B%
```

Als Ergebnis druckt uns dieses Programm nur noch die 50 aus.

Übrigens können bei den Integervariablen Zahlen verwendet werden zwischen minus 32768 und plus 32767, was für unseren Zweck völlig ausreicht. Sollte es nicht ausreichen, bleibt die INT-Funktion, auf die wir weiter oben schon in einem speziellen Zusammenhang eingegangen sind. Die Programmzeile

```
10 A=INT(A)
```

macht auch aus der größten und krummsten Zahl A eine brave ganze Zahl. Also etwa aus 79046.375979 wird 79046.

Zurück zu unserem Problem. Da BASIC immer abrundet, müssen wir in unserem Falle natürlich auch 1000 statt 999 nehmen.

Schauen Sie sich das folgende Programm einfach an. Es enthält keine unbekannten Befehle mehr. Wir werden es im Anschluß Schritt für Schritt durchgehen:

```
20 PRINT "CLR"
30 MAXIMALEZAHL = 999
40 PRINT "Ich denke mir eine Zahl zwischen 0
                    und ";MAXIMALEZAHL;" aus!"
50 ZUFALLSZAHL% = (MAXIMALEZAHL + 1) * RND (1)
60 INPUT "Wie heißt die Zahl"; ZAHL%
70 ANZAHL% = ANZAHL% + 1
80 IF ZUFALLSZAHL% > ZAHL% THEN PRINT "Meine Zahl ist größer!":
   GOTO 60
90 IF ZUFALLSZAHL% < ZAHL% THEN PRINT "Meine Zahl ist kleiner!":
   GOTO 60
100 PRINT "Richtig!"
110 PRINT "Du hast "; ANZAHL%; " Durchläufe benötigt."
```

In Zeile 20 wird, wie üblich, der Bildschirm gelöscht und anschließend die größtmögliche Zahl in einer Variablen abgelegt. Sie erinnern sich doch noch daran, daß auf diese Weise Änderungen am Programm viel leichter zu bewerkstelligen sind. In Zeile 40 wird die Vorgehensweise des Programms beschrieben. Dabei wird in der Zeile auch geschickt die größtmögliche Zahl auf dem Bildschirm ausgegeben. Weil Sie das bestimmt gerne in eigenen Programmen verwenden wollen, wollen wir diese Zeile einmal genauer betrachten. Der PRINT-Befehl gibt drei Dinge auf dem Bildschirm aus:

- den Ausdruck: "Ich denke mir eine Zahl zwischen 0 und "

- den Inhalt der Variablen MAXIMALEZAHL

- und zum Schluß den Ausdruck: " aus!"

Da zwischen den Ausgaben Semikolons stehen, beginnt der Printbefehl keine neue Zeile, sondern gibt alles direkt hintereinander aus. Genausogut hätten wir auch schreiben können:

```
40 PRINT "Ich denke mir eine Zahl zwischen 0 und 999 aus!"
```

Wenn wir dann aber später einmal MAXIMALEZAHL in Zeile 30 auf 9999 setzen, berechnet BASIC Zahlen zwischen 0 und 9999, behauptet aber in der Bildschirmmeldung, nur Zahlen zwischen 0 und 999 raten zu lassen. Von daher ist es besser, den PRINT-Befehl mit dem Variablennamen zu verbinden.

In Zeile 50 wird dann aus der Zufallszahl eine ganze Zahl zwischen 0 und 999 erzeugt. Durch das %-Zeichen weiß BASIC, daß in ZUFALLSZAHL% nur Integerzahlen möglich sind. Durch die Klammer wird erst die Addition ausgeführt und anschließend die Multiplikation, denn auch in BASIC gilt: Punktrechnung geht vor Strichrechnung. Nun erfragen wir durch Zeile 60 den ersten Tip und legen ihn in ZAHL% ab. Da wir ja die Anzahl der Rateversuche mitzählen und am Ende ausgeben lassen wollen, erhöhen wir in Zeile 70 die Integervariable Anzahl% um 1. Dabei benutzen wir einen kleinen Trick. Wenn eine Variable zum ersten Mal benutzt wird und ihr noch kein Wert zugewiesen wurde, hat sie den Wert 0. Daher wird ANZAHL% durch diese Zeile von 0 auf 1 gesetzt und besagt damit, daß wir im 1. Durchlauf sind. Übrigens können Stringvariablen natürlich nicht den Wert 0 haben. Sie werden deshalb als leere Zeichenkette vorgegeben, enthalten also kein einziges Zeichen.

Die folgenden Zeilen stellen die eigentliche Intelligenz des Programms dar. Hier lassen wir BASIC prüfen, ob die Zahl zu groß, zu klein oder richtig geraten war. Beginnen wir mit Zeile 80. Ist die vom Computer berechnete Zahl größer als der Tip, dann wird eine entsprechende Meldung auf dem Bildschirm ausgegeben. Zusätzlich "springt" das Programm zu Zeile 60, wo ein neuer Tip verlangt und die Anzahl der Durchläufe erhöht wird. An dieser Zeile können Sie auch gleich erkennen, daß BASIC mehrere Befehle in einer Zeile erlaubt. Diese werden jeweils durch einen Doppelpunkt getrennt. Nur wenn die Bedingung erfüllt ist, wird der Rest der Zeile 80 weiter bearbeitet, ansonsten übergeht BASIC den Rest der Zeile und macht erst bei 90 weiter.

In Zeile 90 wird nun analog geprüft, ob die Zahl kleiner als der eingegebene Tip ist. Auch dann wird eine entsprechende Meldung ausgegeben und bei Zeile 60 ein neuer Tip geholt. Was ist aber, wenn der Tip weder kleiner noch größer als die Zufallszahl war? Dann müssen beide Zahlen gleich sein. In Zeile 100 gibt BASIC eine Erfolgsmeldung aus und schreibt anschließend in Zeile 110 die Anzahl der Durchläufe auf den Bildschirm. Dabei wird wieder eine Bildschirmausgabe aus festen Texten und

einer Variablen zusammengesetzt, wie wir es schon in Zeile 40 benutzt haben.

5.3 Dritte Anwendung der neuen Befehle: Das große Einmaleins

Bevor wir uns neuen interessanten Möglichkeiten von BASIC zuwenden, wollen wir das Programm "Zahlenraten" noch so abändern, daß man damit das große Einmaleins üben kann. Sie werden sicherlich eine sinnvolle Anwendung dafür finden. Zu überlegen wäre zum Beispiel, ob Sie Ihren C64 als Partner beim Nachhilfeunterricht einsetzen. Dann kommt das Geld bald wieder herein.

Das folgende Programm benutzt praktisch nur dieselben Befehle wie das vorherige. Deshalb brauchen wir es auch nur kurz zu erklären:

Daß wir jetzt die Variable MAXWERT statt MAXIMALWERT verwenden, hängt damit zusammen, daß uns das Tippen zu langwierig vorkam. Sie können gerne noch kürzere Namen wählen, sollten aber sicherstellen, daß diese Bezeichnungen aussagekräftig genug bleiben, so daß man das Programm schnell versteht! Oder aber Sie sind ganz radikal und nehmen nur die beiden Buchstaben, die der C64 zu unterscheiden vermag.

```
20 MAXWERT = 19
30 PRINT "CLR"
40 PRINT "            Das große Einmaleins              "
50 PRINT
60 ZAHL1% =(MAXWERT + 1) * RND(1)
70 ZAHL2% =(MAXWERT + 1) * RND(1)
80 PRINT "Wieviel ergibt "; ZAHL1%; " * "; ZAHL2%
90 INPUT "Ergebnis "; ERG%
100 COMPUTERERG% = ZAHL1% * ZAHL2%
110 IF COMPUTERERG% = ERG% THEN PRINT "Richtig!!!":
                           RICHTIG%=RICHTIG%+1:GOTO 130
120 PRINT "Falsch!!!": FALSCH% = FALSCH% + 1
130 INPUT "Möchtest Du noch eine Aufgabe (j/n)"; EINGABE$
140 IF EINGABE$ = "j" THEN GOTO 30
150 PRINT "Richtig:";RICHTIG%, "Falsch:";FALSCH%
```

Programme stückweise auf dem Bildschirm

Übrigens: Sie müssen sich nicht immer ein ganzes Programm mit LIST auf dem Bildschirm ausgeben lassen. Sobald das Programm mehr Zeilen umfaßt, als auf den Bildschirm passen, geht das sowieso nicht mehr. Daher bietet BASIC noch speziellere Möglichkeiten:

Wird nach LIST eine Zeilennummer angegeben, so wird nur diese Zeile angezeigt. LIST 100 - 200 zeigt entsprechend alle Programmzeilen in den angegebenen Grenzen.

Erklärung des Programms

In Zeile 20 wird der MAXWERT für die erzeugten Zufallszahlen auf 19 gesetzt, entsprechend dem großen Einmaleins. Wollen Sie nur das kleine Einmaleins üben lassen, so tragen Sie hier einfach eine 9 ein. Die Zeilen 30 bis 50 sorgen für einen gelöschten Bildschirm mit einer Überschrift. In den Zeilen 60 und 70 werden zwei zufällige, ganze (%) Zahlen zwischen 0 und MAXWERT erzeugt. In Zeile 80 wird dann die Aufgabenstellung wieder aus festen Texten und Variablen zusammengesetzt.

Nach der Abfrage des Ergebnisses in Zeile 90 wird vom Rechner das (hoffentlich) richtige Ergebnis berechnet und in Zeile 110 mit der eingegebenen Zahl verglichen. Sind beide Zahlen gleich, wird eine entsprechende Meldung auf den Bildschirm geschrieben, die Zählvariable RICHTIG% um 1 erhöht und zu Zeile 130 gesprungen.

Ansonsten muß die eingegebene Zahl falsch sein. Also wird in Zeile 120 "Falsch!!!" auf den Bildschirm geschrieben und die Zählvariable FALSCH% um 1 erhöht. In jedem Fall wird in Zeile 130 nachgefragt, ob noch eine Aufgabe gestellt werden soll, und bei "j" zu Zeile 30 gesprungen. Andernfalls wird die Anzahl der richtigen (RICHTIG%) und der falschen (FALSCH%) Lösungen angegeben und das Programm beendet.

5.4 Falls mal kein Würfel da ist: Kein Problem mit BASIC

Nach so viel ernsthaften Anwendungen sollten wir uns noch ein wenig Spaß gönnen und uns einem bekannten Spiel zuwenden. Keine Angst, jetzt wird nicht mehr lange programmiert. Aber stellen Sie sich einmal vor, Sie wollen das Angenehme mit dem Nützlichen verbinden und Ihren C64 für sich und die ganze Familie würfeln lassen, etwa bei dem bekannten Spiel: "Mensch ärgere Dich nicht". Dann können Sie das folgende kurze Programm laufen lassen (CLR entspricht dabei der Tastenkombination Shift+CLR/Home):

```
10 PRINT "CLR"
20 ZUFALLSZAHL = INT (6*RND(1))+1
30 PRINT "DU HAST EINE ";ZUFALLSZAHL;" GEWUERFELT!"
40 INPUT "NOCH EIN WURF";EINGABE$
50 IF EINGABE$ = "J" THEN GOTO 20
```

Später, wenn das Vertrauen in die Computer in Ihrer Familie Ihnen unheimlich werden sollte, können Sie es leicht wieder zerstören, indem Sie an dem Programm so lange herummanipulieren, bis nach Eingabe Ihres Namens häufiger ein höherer Wert herauskommt als bei den anderen! Dann werden Sie zwar wahrscheinlich wieder zu dem nicht so leicht beeinflußbaren Würfel zurückkehren, aber das Programmieren in BASIC, das beherrschen Sie. Und das hatten wir Ihnen versprochen!

5.5 Zusammenfassung

Nach den Befehlen GOTO, IF THEN und FOR NEXT haben wir in diesem Kapitel noch das folgende kennengelernt:

1. Mit dem Befehl REM (für englisch: REMARK = Bemerkung) können Sie BASIC mitteilen, daß das, was dahinter steht, nur für Sie eine Hilfe sein soll und keinen Befehlscharakter hat. Sinnvoll sind Bemerkungen, um Hinweise zu Programmteilen zu geben: REM HAUPTPROGRAMM.

2. Mit STEP können Sie bei einem FOR-NEXT-Befehl eine Schrittweite angeben: FOR LITER = 1 to 30 STEP .2 bedeutet, daß immer um 0.2 erhöht wird. STEP kann auch

negativ sein und bedeutet dann, daß von einem höheren Wert rückwärts bis zu einem anderen gezählt wird.

3. Der LIST-Befehl kann durch Angabe von Zeilennummern und Programmbereichen differenziert und gezielt angewendet werden:

   LIST 50 bringt nur die Programmzeile 50 auf den Bildschirm, LIST 50 - 100 stellt den Programmbereich zwischen Zeile 50 und Zeile 100 zur Verfügung.

4. Variablen mit einem Prozentzeichen am Schluß (LITER%) nennt man Integervariablen. Sie verstehen alle ihnen zugewiesenen Zahlenwerte als ganze Zahlen, dabei können diese Werte von -32767 bis +32768 reichen. Die Zuweisung

```
KOSTEN% = 34.75
```

   bedeutet also, daß das Programm diese Variable immer mit dem Wert 34 füllt und die Nachkommastellen entfallen läßt. Neben dem Prozentzeichen (%) als Anhängsel beim Variablennamen kann man auch - und das ohne die oben genannte Beschränkung - die INT-Funktion verwenden. In der Programmzeile

```
10 A=INT(A)
```

   wird aus jeder Zahl A die entsprechende ganze Zahl gemacht (durch Abrunden der Folgestellen). Statt A kann in der Klammer natürlich auch eine Formel mit anderen Variablen stehen, der Ergebnis dann zur ganzen Zahl gemacht wird, etwa:

```
10 A=INT(B*C/2)
```

5. RND-Funktion und INT-Funktion.

   RND(X) ist eine Funktion in BASIC, mit deren Hilfe Sie Zufallszahlen produzieren lassen können. Dabei reicht es, immer wieder für X eine 1 zu setzen. Die Zahlen sind dann immer unterschiedlich. Als Ergebnis werden immer Werte zwischen 0 und 1 produziert. Will man etwa ganzzahlige Werte haben, kann man das einfach mit Hilfe der

INT-Funktion erreichen. INT (3.6) ist zum Beispiel 3. Allgemein kann man ganze Zahlen durch das folgende kurze Programm erreichen:

```
10 ZUFALLSZAHL = INT(10*RND(1))
```

6. Einfache Grafik und andere nützliche Dinge

Nachdem wir im letzten Kapitel viel gerechnet haben, wollen wir Ihnen in diesem Kapitel zeigen, wie man Zahlen einfach auf dem Bildschirm darstellen kann. Das Endergebnis wird unter anderem eine sogenannte Balkengrafik sein, die Sie sicherlich aus den Wahlsendungen kennen.

6.1 Grafik ganz einfach: Die ASCII-Grafik

Sicherlich haben Sie schon von den Grafikfähigkeiten des C64 gehört. Als er von Commodore herausgebracht wurde, waren diese Fähigkeiten in seiner Preisklasse einmalig. Allerdings stellte sich bald heraus, daß diese Grafik nicht ganz einfach zu nutzen ist und es auch keine BASIC-Befehle gibt, die den Zugang zu den Grafikqualitäten einfach ermöglichen. Dies wurde erst später mit dem großen Bruder des C64, dem C128, und dessen starkem BASIC V7 erreicht. Wegen der eben beschriebenen Schwierigkeiten wollen wir uns mit der sogenannten hochauflösenden Grafik erst im Kapitel 8 beschäftigen und Ihnen hier lieber zeigen, wie Sie auf einfache Weise etwas grafisch auf Ihrem Bildschirm darstellen können. Dazu benutzen wir einfach die Zeichen, die Ihr C64 schon fest eingebaut hat.

Diese Zeichen heißen ASCII-Zeichen. ASCII ist die Abkürzung für "American Standard Code for Information Interchange", d.h. für genaue Absprachen darüber, welche Zahl der Computer einem Zeichen zuordnet. Denn unser C64 kann natürlich nur in Zahlen "denken" und muß jedes über die Tastatur oder sonstwie an ihn übergebene Zeichen erst in eine solche Zahl übersetzen. Die Ziffern beginnen zum Beispiel bei der Nummer 48 (= 0), die Buchstaben bei der Nummer 65 (= a).

Allerdings ist es mit dem Standard bei ASCII nicht sehr weit her. Dieser Standard legt nämlich nur die Zeichen für den Zahlenbereich von 0 - 127 fest, eine Speicherstelle kann aber Werte von 0 - 255 aufnehmen. So haben viele Computerhersteller für

die Zahlen von 128 - 255 willkürlich weitere Zeichen festgelegt. Das wurde schon dadurch unterstützt, daß der "amerikanische Standard" ASCII natürlich keine Umlaute vorsieht und deutsche Computerhersteller notgedrungen die Umlaute auf nicht standardisierte Zahlen legen mußten.

Commodore ging hier sogar noch etwas weiter und hielt sich auch im Bereich von 0 - 127 nicht genau an die Zahlenvorgaben. Trotzdem wollen wir im Zusammenhang mit diesen Zeichen von ASCII-Zeichen sprechen, weil das Grundprinzip bei den meisten Computern gleich ist.

Ein Zeichen wird zur Zahl: ASCII-Werte auf dem Bildschirm

Bevor wir nun die ersten grafischen Versuche machen, wollen wir Ihnen zeigen, wie Sie an den Zusammenhang zwischen den Zeichen und ihren Zahlen herankommen. Zum einen gibt es da in Ihrem Computerhandbuch im Anhang eine Tabelle. Oft ist es aber einfacher, mit einem kleinen Programm die Zahlen für die Zeichen herauszufinden und dabei gleich zwei neue Befehle kennenzulernen. Geben Sie einmal das folgende Programm ein. Bevor Sie es mit RUN starten, drücken Sie bitte einmal gleichzeitig die C=-Taste mit der Shift-Taste, so daß nun kleine Buchstaben auf dem Bildschirm erscheinen und große nur, wenn gleichzeitig die Shift-Taste gedrückt wird.

```
10 GET A$
20 IF A$ = "" THEN 10
30 PRINT A$,ASC(A$)
40 GOTO 10
```

Wenn Sie jetzt eine Taste drücken, zeigt der C64 zuerst das zugehörige Zeichen und dann die Zahl für dieses Zeichen an. Bei einem a ist das eine 65 und bei einem Z eine 218. Besonders interessant wird das bei Tasten, die normalerweise kein Zeichen erzeugen oder sogar eine Funktion auslösen: Das Leerzeichen hat die Zahl 32, die Funktionstaste F1 ergibt 133, und wenn Sie Shift+Clr/Home drücken, wird zwar der Bildschirm gelöscht, aber zusätzlich auch die zugehörige Zahl - eine 147 - angezeigt.

Wie funktioniert unser kleines Programm? Zuerst fragt der Befehl GET A$ in Zeile 10, ob eine Taste gedrückt wurde. Die Zeile 20 verhindert eine Fehlermeldung, falls keine Taste gedrückt wurde. Dann liegt kein Zeichen vor - das erkennt der C64 an den Anführungsstrichen ohne Inhalt - und in Zeile 30 würde der Befehl ASC(A$) eine Fehlermeldung erzeugen. Ist aber ein Zeichen vorhanden, dann wandelt der Befehl ASC(A$) es in die zugehörige Zahl um. In Zeile 40 beginnt der Vorgang erneut.

Wenn Sie übrigens verhindern möchten, daß die zugehörige Funktion einer Taste auch ausgeführt wird, ändern Sie einfach Zeile 30 um in

```
30 PRINT ASC(A$)
```

Dann wird bei Shift+Clr/Home nur noch die 147 angezeigt, nicht mehr aber der Bildschirm gelöscht. Unser Programm läuft natürlich solange weiter, bis wir die RUN/STOP-Taste drücken.

Und nun umgekehrt: Eine Zahl wird zum Zeichen

Das Gegenstück zum Befehl ASC bildet der Befehl CHR$, er wandelt eine Zahl zwischen 0 und 255 in das zugehörige Zeichen um. Das folgende kleine Programm zeigt die Funktion dieses Befehls:

```
10 INPUT "BITTE EINE ZAHL EINGEBEN "; A
20 IF A < 0 OR A > 255 THEN GOTO 10
30 PRINT CHR$(A)
40 GOTO 10
```

Probieren Sie ruhig verschiedene Zahlen aus, passieren kann ja wegen der Sicherheitsabfrage in Zeile 20 nichts. Besonders interessant sind hier wieder die Zahlen, die für eine Taste mit einer besonderen Funktion stehen: 147 für Bildschirmlöschen, 19 für Cursor Home und so weiter. Um dieses Programm zu unterbrechen, müssen Sie sofort nach der Return-Taste die RUN/STOP-Taste drücken. Sollte Ihnen das nicht gelingen, dann drücken Sie gleichzeitig die CTRL-Taste und die RESTORE-Taste.

Die erste Grafik: Ein Dreieck

Nun wollen wir aber endlich die erste Grafik erzeugen. Schalten Sie jetzt aber bitte wieder durch C= und Shift auf die normale Zeichendarstellung zurück, da wir sie nachher benötigen.

Eine einfache Möglichkeit für ASCII-Grafik stellt das nun folgende Programm dar:

```
10 PRINT CHR$(147)
20 A$ = "*"
30 FOR I = 1 TO 20
40 A$ = A$ + "*"
50 PRINT A$
60 NEXT I
```

Das Programm produziert ein mit Sternen gefülltes Dreieck auf dem Bildschirm. Auf diese Weise kann man eine ganze Menge schöner Effekte erreichen.

6.2 Sternenhimmel

Beginnen wir zuerst mit einem Programm, das schon etwas schwieriger ist und auf dem Bildschirm einen Sternenhimmel erzeugt.

Wenn Sie zu den Hobbyastronomen gehören, sollten Sie jetzt allerdings kein Hilfsmittel für Ihre abendlichen Himmelserkundungen erwarten. Wir wollen vielmehr spielerisch auf dem Bildschirm Sternkonfigurationen erzeugen, die echt sein könnten.

Wenn Sie uns dann durch dies Kapitel gefolgt sind, werden Sie sicherlich auch eine Möglichkeit finden, echte Sternbilder Ihrer Wahl auf den Bildschirm zu bekommen.

Auf jeden Fall müssen wir natürlich erst einmal wissen, wie man überhaupt einen einzigen Stern auf den Bildschirm-Himmel bekommt. Glücklicherweise sieht das eben bereits verwendete Multiplikationszeichen "*" einem Stern schon recht ähnlich.

Auf den Bildschirm können wir es mit dem PRINT-Befehl schreiben. Probieren Sie einmal das folgende Programm:

```
10 PRINT CHR$(147);
20 FOR I = 1 TO 20
30 X = 40 * RND(1)
40 Y = 25 * RND(1)
50 FOR J = 1 TO Y
60 PRINT CHR$(17);
70 NEXT J
80 FOR J = 1 TO X
90 PRINT CHR$(29);
100 NEXT J
110 PRINT "*";
120 PRINT CHR$(19);
130 NEXT I
```

Erklärung des Programms

Zuerst wird der Bildschirm gelöscht. Anschließend dient die I-Schleife (äußere Schleife: Zeile 20 bis 130) dazu, 20 Sterne zu erzeugen. In Zeile 30 und 40 werden zufällige Werte für die Position erzeugt, diese dürfen zwischen 0 und 39 für die waagerechte Position und zwischen 0 und 24 für die senkrechte liegen, dafür sorgen die beiden Zeilen. Nun benutzen wir einen einfachen Trick, um den Cursor für die Ausgabe des Zeichens an die gewünschte Position zu bekommen: Für die Y-Position geben wir solange Cursor Down (CHR$(17)) auf dem Bildschirm aus, bis die Stelle erreicht ist, und für die X-Position anschließend Cursor rechts (CHR$(29). Wichtig ist dabei das abschließende Semikolon, damit der C64 nicht den Cursor an den Anfang der nächsten Zeile setzt. Wenn der Cursor nun an der richtigen Stelle steht (nach Abarbeitung der zwei J-Schleifen), geben wir den Stern auf dem Bildschirm aus. Wichtig ist, daß wir anschließend in Zeile 120 den Cursor wieder in die obere linke Ecke setzen (CHR$(19) = Cursor Home), da wir die Position ja immer von oben links rechnen.

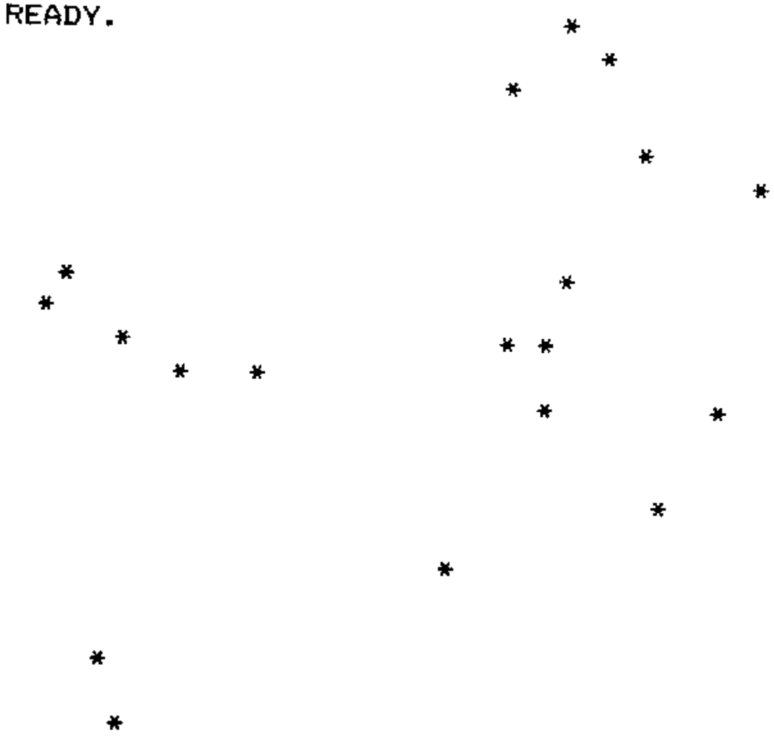


Bild 6-1: Der Sternhimmel im ... - Ein Produkt des Zufalls

Da wir in unseren folgenden Programmen immer wieder ein Zeichen an eine beliebige Stelle des Bildschirms setzen wollen, verlassen wir an dieser Stelle für einen Augenblick die Welt der Grafik und zeigen Ihnen eine interessante Möglichkeit, sich viel Schreibarbeit zu ersparen: die Unterprogramme. Vielleicht haben Sie aber vorher Lust, sich noch die eine oder andere Sternkarte auf dem Bildschirm anzeigen zu lassen. Dabei sollten Sie auch einmal die Anzahl der Sterne in Zeile 20 ändern. Denn hier gab es schon unter den Autoren Unstimmigkeiten, bei wie vielen Sternen der Himmel am schönsten aussieht.

6.3 Unterprogramme: GOSUB und RETURN

Unser Problem besteht darin, daß wir in den folgenden Programmen an verschiedenen Stellen ein Zeichen auf dem Bildschirm ausgeben müssen. Nun wollen wir natürlich nicht jedesmal die 8 Programmzeilen hinschreiben, die das erledigen. Da wäre es doch schön, wenn wir diese nur einmal hinschreiben und dann mehrmals nutzen könnten. Mit den bisher verwendeten Befehlen funktioniert das aber nicht, wie das folgende Programmbeispiel zeigt:

```
10 X = 10: Y = 20
20 GOTO 100
30 X = 30: Y = 10
40 GOTO 100
50 X = 5: Y = 15
60 GOTO 100
70 END
100 REM UNTERPROGRAMM, STELLVERTRETEND FUER STERNENAUSGABE
110 PRINT X,Y
120 GOTO 30
```

Der Befehl END sorgt übrigens dafür, daß ein Programm nicht bis zur letzten vorhandenen Zeile abgearbeitet wird, sondern sofort endet.

Beim erstenmal funktioniert das Programm richtig. Es werden 10 und 20 ausgegeben, und anschließend wird in Zeile 30 weitergemacht. Aber schon nach der zweiten Ausgabe springt das Programm immer wieder zu Zeile 30 und erreicht nie den Programmteil ab Zeile 50. Das Problem liegt also darin, daß der Rücksprung aus der Ausgabe immer zur selben Zeilennummer erfolgt. Wir brauchen also eine Möglichkeit, von einer beliebigen Stelle im Programm zu einer häufig benötigten Stelle zu springen und anschließend an die alte Stelle zurückzukehren. Dieses Problem lösen sogenannte Unterprogramme zusammen mit den Befehlen GOSUB und RETURN.

GOSUB ... RETURN: Viele Wege führen ins Unterprogramm

Ein Unterprogramm ist ein Teil eines Programms, der mit RETURN abgeschlossen wird. Nun springt man die erste Zeile dieses Programmabschnitts nicht mit GOTO, sondern GOSUB an. Dadurch merkt sich der C64 die augenblickliche Stelle (gewissermaßen die Absprungstelle), macht an der angegebenen Zeilennummer weiter und kehrt bei dem GOSUB wieder zu gemerkten Stelle zurück. Nehmen wir dazu das obige Beispiel:

```
10 X = 10: Y = 20
20 GOSUB 100
30 X = 30: Y = 10
40 GOSUB 100
50 X = 5: Y = 15
60 GOSUB 100
70 END
100 REM UNTERPROGRAMM, STELLVERTRETEND FUER STERNENAUSGABE
110 PRINT X,Y
120 RETURN
```

Das Unterprogramm beginnt in Zeile 100 und endet in Zeile 120 mit dem RETURN. Jetzt wird es insgesamt dreimal ausgeführt. Wichtig ist das END in Zeile 70. Würde das Programm nicht beendet, so würde es ein weiteres Mal das Unterprogramm ab Zeile 100 abarbeiten und am Schluß auf das RETURN treffen. Da kein Aufruf des Unterprogramms mit GOSUB erfolgte, würde sich BASIC mit "RETURN WITHOUT GOSUB" beschweren.

Ein Unterprogramm für viele Fälle: Die Ausgabe eines Zeichens

Mit diesen Unterprogrammen können wir nun unser Problem elegant lösen. Wir schreiben die Zeilen für die Ausgabe eines Zeichens als Unterprogramm und nutzen es in den folgenden Programmen. Das sieht dann folgendermaßen aus:

```
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
```

```
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29);
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

Speichern Sie dieses Unterprogramm bitte unter dem Namen UPZEICHEN auf Diskette ab. Wenn Sie nun in einem Programm Zeichen an beliebiger Stelle ausgeben wollen, laden Sie zuerst dieses Programm und erweitern es dann einfach um den Rest. Um ein Zeichen auszugeben, müssen Sie X und Y auf die richtigen Werte setzen und in X$ das Zeichen haben, das ausgegeben werden soll. Aber Achtung: Das Programm benutzt J als Schleifenzähler. Wenn Sie also in Ihrem Programm an anderer Stelle die Variable J verwenden, wird sie durch das Unterprogramm verändert.

Eine Besonderheit des Programms steckt noch in den Zeilen 10005 und 10040. Die jeweiligen Schleifen dahinter werden nämlich in jedem Fall einmal abgearbeitet, auch wenn der Wert zu Beginn kleiner als der Endwert ist (also FOR J = 1 TO 0). Dadurch kann aber kein Zeichen in der oberen linken Ecke ausgegeben werden. Deshalb überspringen wir bei 0 die jeweilige Schleife.

6.4 Malen mit dem Computer

Sicherlich haben Sie schon einmal Computerbilder gesehen, die mit einem speziellen Malprogramm erstellt wurden. Unabhängig davon, ob Sie diese Bilder hübsch oder häßlich fanden - es ist viel Arbeit, solche Bilder zu erstellen. Daher wollen wir Ihnen hier ein kleines Programm vorstellen, bei dem der Computer selbst malt und Sie keinen Finger zu rühren brauchen. Dieses Programm soll Ihnen gleichzeitig einen zweiten Weg zeigen, Grafik auf dem Bildschirm zu erstellen. Bei unserem letzten Programm hatte die Position eines Sterns nichts mit der Position des vorhergehenden zu tun. In unserem neuen Programm wollen wir aber von einem Punkt ausgehend weitermalen, also einen Punkt jeweils neben seinen Vorgänger setzen.

Der C64 malt - zunächst mit einem Stift

Da ein C64 keine Vorstellung davon hat, wie eine MONA LISA aussehen sollte, wollen wir die Richtung, in der der nächste Punkt gesetzt wird, zufällig bestimmen lassen. Daher wird unser Programm etwa folgendermaßen ablaufen: Wir setzen an einer Stelle des Bildschirms einen Punkt und merken uns seine Koordinaten. Anschließend entscheidet der Zufall, ob der nächste Punkt links, rechts, oben oder unten gesetzt wird. Wir verändern die Koordinaten entsprechend und setzen den neuen Punkt. Dann beginnt der Vorgang erneut, eine ideale Voraussetzung für eine Schleife.

Es bleibt eigentlich nur noch die Frage, wie wir zufällig eine Richtung bestimmen können. Da wir vier mögliche Richtungen haben, bestimmen wir zufällig eine Zahl zwischen 1 und 4 und nehmen für jede Zahl eine der Richtungen.

Auf eines müssen wir aber wieder achten: Wir dürfen nur erlaubte Koordinaten nehmen. Wenn wir also die Koordinaten geändert haben, müssen wir noch prüfen, ob sie im erlaubten Bereich liegen. Andernfalls müssen wir sie so ändern, daß sie wieder im richtigen Bereich liegen und trotzdem das Zeichnen an einer passenden Stelle weitergeht. Nehmen wir ein Beispiel:

In X ist die X-Koordinate gespeichert, also die Spalte. Im Augenblick hat X den Wert 39, und damit ist der letzte Punkt am rechten Bildschirmrand gesetzt worden. Nun wird durch Zufall als neue Richtung "rechts" bestimmt. Wir müssen nun die X-Koordinate durch "X = X + 1" um eins erhöhen. Dadurch erhält X aber den Wert 40, und der ist nicht erlaubt. Wir wollen einfach festlegen, daß, wenn ein Punkt rechts aus dem Bildschirm verschwindet, er von links wieder hereinkommt. Dazu brauchen wir in unserem Fall einfach nur X auf 0 zu setzen.

Mit diesen Überlegungen sieht unser Programm dann folgendermaßen aus. Denken Sie bitte daran, vor der Eingabe das Unterprogramm ab Zeile 10000 mit LOAD "Upzeichen",8 von Diskette zu laden, damit Sie die letzten Zeilen nicht eingeben müssen.

```
10 PRINT CHR$(147)
15 X$ = "*"
20 X = 20
30 Y = 12
40 FOR I = 1 TO 1000
50 R% = 4 * RND(1) + 1
60 IF R% = 1 THEN X = X + 1
70 IF R% = 2 THEN X = X - 1
80 IF R% = 3 THEN Y = Y - 1
90 IF R% = 4 THEN Y = Y + 1
100 IF X = -1 THEN X = 39
110 IF X = 40 THEN X = 0
120 IF Y = -1  THEN Y = 24
130 IF Y = 25 THEN Y = 0
140 GOSUB 10000
150 NEXT I
160 END
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

Erklärung des Programms

In Zeile 20-30 setzen wir die Startkoordinaten auf die Mitte des Bildschirms. In der nächsten Zeile beginnen wir eine Schleife, die 1000mal durchlaufen wird. Dadurch wird der Bildschirm nicht zu voll und bleibt nicht zu leer. In Zeile 50 wird dann eine zufällige ganze Zahl R% zwischen 1 und 4 für die Richtung bestimmt. In den 4 folgenden Zeilen wird nun für jede mögliche Richtung die entsprechende Änderung der Koordinaten vorgenommen. Bei "rechts" wird die X-Koordinate um 1 erhöht, bei "links" um eins vermindert. Dasselbe geschieht analog mit der Y-Koordinate. Für "unten" wird sie erhöht, für "oben" vermindert.

Sie können übrigens viel Abtipp-Arbeit sparen, wenn Sie ähnlich aussehende Zeilen einfach verdoppeln und dann ändern. Nehmen wir als Beispiel die Zeilen 60 bis 90. Sie haben die Zeile 60 hingeschrieben und die Eingabetaste betätigt. Ändern Sie nun einfach die Zeilennummer in 70, und drücken Sie erneut die Eingabetaste. Dasselbe machen Sie dann noch mit 80 und 90. Wenn Sie sich Ihr augenblickliches Programm nun mit LIST anzeigen lassen, sieht es folgendermaßen aus:

```
10 PRINT CHR$(147)
15 X$ = "*"
20 X = 20
30 Y = 12
40 FOR I = 1 TO 1000
50 R% = 4 * RND(1) + 1
60 IF R% = 1 THEN X = X + 1
70 IF R% = 1 THEN X = X + 1
80 IF R% = 1 THEN X = X + 1
90 IF R% = 1 THEN X = X + 1
```

Nun müssen Sie in Zeile 70 nur noch "R% = 1" in "R% = 2" und "X + 1" in "X - 1" ändern (am Schluß Eingabetaste bitte nicht vergessen). Genauso verfahren Sie wieder in den Zeilen 80 und 90. Kontrollieren Sie zum Schluß mit LIST noch einmal die Änderungen, und Sie haben eine ganze Menge Arbeit gespart.

Nachdem nun die neue Richtung festgelegt ist und die Koordinaten entsprechend geändert wurden, muß natürlich noch die Prüfung auf den erlaubten Bereich erfolgen. Wie vorher an einem Beispiel erklärt, werden nun in den Zeilen 100 bis 130 beide Koordinaten daraufhin geprüft, ob sie zu groß oder zu klein sind, und gegebenenfalls geändert. In der Zeile 140 wird dann unser neues Unterprogramm aufgerufen, das den Stern an der aktuellen Stelle auf den Bildschirm schreibt. In der letzten Zeile ist die Schleife zu Ende, und nach 1000 Durchläufen endet das Programm.

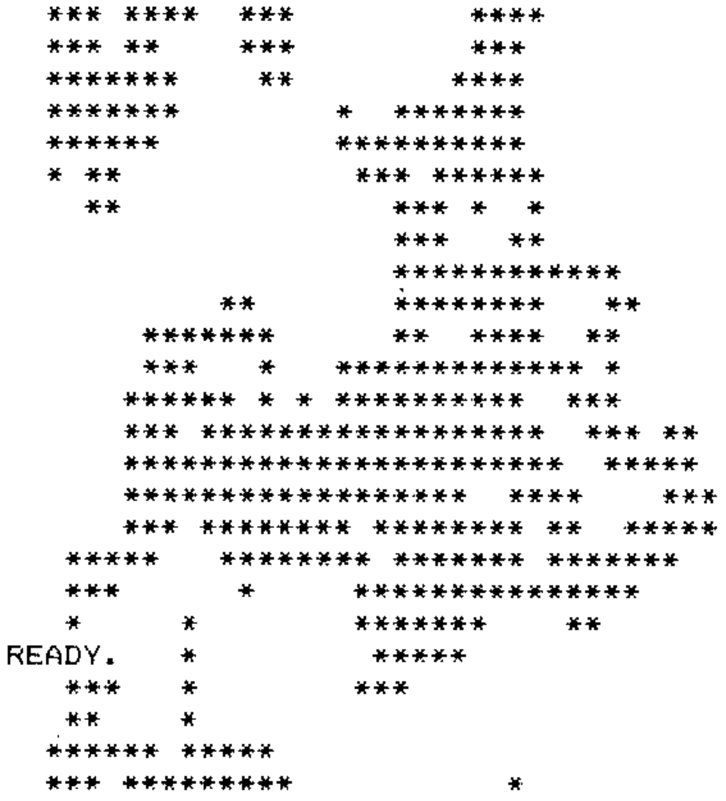


Bild 6-2: Der C64 beim Malen und ein mögliches Ergebnis

Wenn Sie das Programm gespeichert haben (SAVE "MALEN",8) und starten, können Sie sich entspannt im Sessel zurücklehnen und dem Computer bei der Malarbeit zusehen. Wenn er nach 1000 Durchläufen fertig ist, erscheint irgendwo am linken Bildschirmrand das READY mit dem blinkenden Cursor. Vielleicht haben Sie sich schon manchmal darüber gewundert, daß am Ende unserer bisherigen Grafik-Programme manchmal bei der Eingabe eines neuen Befehls ein SYNTAX ERROR erscheint. Das liegt daran, daß in der Zeile, in der Sie den Befehl eingeben, noch irgendwelche Sternchen auf dem Bildschirm stehen, die der C64 natürlich nicht als gültigen Befehl anerkennt. Am

besten löschen Sie einfach den Bildschirm durch Shift+Clr/Home, bevor Sie einen Befehl eingeben.

Der C64 malt mit wechselnden Stiften

Wir wollen das Programm noch um eine interessante Möglichkeit erweitern und Ihnen dann auch gleich zeigen, wie Sie solche Programme beenden können, ohne daß anschließend der Bildschirm voller Sterne bleibt. Vielleicht ist Ihnen aufgefallen, daß der Rechner manchmal sekundenlang nicht zu malen scheint. Das liegt daran, daß zufällig neue Punkte in einen Bereich geschrieben werden, der schon voller Punkte ist. Unser Künstler (der C64) besitzt ja auch nur einen einzigen Stift zum Zeichnen. Dem wollen wir abhelfen. Deshalb wollen wir das Programm so abändern, daß Sie den Stift jederzeit wechseln können.

Das ist mit einer einzigen, trickreichen Zeile möglich. Fügen Sie bitte in das Programm folgende Zeilen ein:

```
135 GET A$: IF A$ <> "" THEN X$ = A$

160 GET A$: IF A$ = "" THEN 160
170 PRINT CHR$(147)
180 END
```

Nun können Sie, während der C64 malt, einfach eine Taste drücken, und dieses Zeichen wird anschließend zum Malen benutzt. Besonders interessant ist dadurch auch die Möglichkeit, durch Drücken der Leertaste wieder Zeichen vom Bildschirm zu löschen und durch Druck auf Shift+Clr/Home und anschließend "*" den Bildschirm zu löschen und dann wieder mit dem Stern zu beginnen.

Wir haben hier mal einiges ausprobiert und zeigen Ihnen einfach das Ergebnis. Ihrer eigenen Phantasie sind natürlich keine Grenzen gesetzt. Und beim Betrachten Ihres C64-Künstlers erholen Sie sich dann gleichzeitig für unsere weiteren gemeinsamen Erfahrungen.

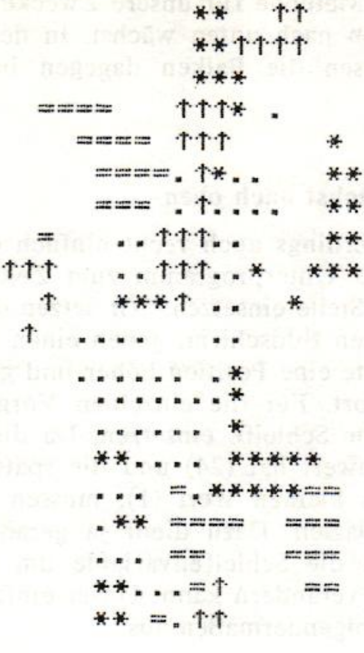


Bild 6-3: Der C64 malt mit wechselnden Stiften

6.5 Balken

Nachdem wir nun schon viel Übung darin haben, Grafiken dadurch zu erstellen, daß wir Zeichen an beliebigen Stellen auf den Bildschirm schreiben, wollen wir uns langsam an das erste große Projekt wagen: die Balkengrafik. Dazu überlegen wir zuerst einmal, wie wir am besten einen Balken auf dem Bildschirm erzeugen können. Eine ganz einfache Lösung wäre sicherlich:

```
10 PRINT CHR$(147);
20 FOR I = 1 TO 20
30 PRINT "     ***"
40 NEXT I
```

Allerdings ist diese Methode für unsere Zwecke ungeeignet, weil der Balken von oben nach unten wächst. In den professionellen Darstellungen wachsen die Balken dagegen immer von unten nach oben.

Der erste Balken wächst nach oben

Das können wir allerdings auch recht einfach erreichen, indem wir unser bewährtes Unterprogramm zum Zeichnen eines Zeichens an beliebiger Stelle einsetzen. Wir setzen die Y-Koordinate einfach unten auf den Bildschirm, geben einen Balkenstrich aus, setzen die Koordinate eine Position höher und geben wieder eine Zeile aus und so fort. Für die einzelnen Vorgänge wollen wir natürlich wieder eine Schleife einsetzen. Da die tiefste Position einen großen Zeilenwert hat (24) und die später zu zeichnende Position oben einen kleinen Wert (1), müssen wir die Schleife "rückwärts" laufen lassen. Dazu dient ja gerade der STEP-Befehl, mit dem man die Schleifenvariable um einen beliebigen Wert (also auch -1) verändern kann. Unser einfaches Balkenprogramm sieht dann folgendermaßen aus:

```
10 PRINT CHR$(147)
15 X$ = "***": X = 1
20 FOR I = 24 TO 1 STEP -1
30 Y = I
40 GOSUB 10000
50 NEXT I
60 GET A$
70 IF A$ = "" THEN 60
80 PRINT CHR$(147)
90 END
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

Mit diesem Programm wächst der Balken schön von unten nach oben. Wenn Ihnen das Ganze zu wenig nach Balken aussehen sollte, ersetzen Sie doch einfach in der Zeile 15 die drei Sternchen durch das Zeichen, das sich ergibt, wenn Sie die C=-Taste (Commodore-Taste) gedrückt halten und zusätzlich die +-Taste rechts neben der Null-Taste betätigen!

```
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
***
```

Bild 6-4: Der erste Balken erscheint auf dem Bildschirm

Allerdings ist ein Balken natürlich noch keine Balkengrafik. Erst durch unterschiedlich hohe Balken bekommt eine Balkengrafik ja ihren Aussagewert. Daher wollen wir als nächstes versuchen, unser Programm so abzuändern, daß mehrere unterschiedlich hohe Balken auf den Bildschirm gezeichnet werden.

Mehrere Balken nebeneinander

Dazu benötigen wir natürlich erst einmal unterschiedliche Zahlen für die Höhe der einzelnen Balken. Diese wollen wir wieder mit dem RND-Befehl erzeugen. Zusätzlich müssen wir darauf achten, daß jeder neue Balken ein Stück weiter rechts gezeichnet wird. Unser Programm muß also folgendes machen:

1. Bildschirm löschen und Zufallszahlen festlegen.
2. Die folgenden Schritte (Nr. 3 - 5) für jeden einzelnen Balken durchführen.
3. Aus einer Zufallszahl die Höhe berechnen.
4. Zeile für Zeile bis zur gewünschten Höhe zeichnen.
5. Nach dem Zeichnen aller Teile für alle Balken auf eine Taste warten und den Bildschirm löschen.

Exkurs: Kleines Einmaleins mit Hilfe verschachtelter Schleifen

In unserer Aufstellung fallen zwei Stellen auf, an denen jeweils etwas mehrfach getan werden muß. Dies ist einmal der Punkt 2 und das Zeichnen in Punkt 4. Deshalb wollen wir an beiden Stellen eine Schleife einsetzen. Allerdings liegt die zweite Schleife (4) innerhalb der ersten Schleife (2). Das nennt man auch verschachtelte Schleifen. Wir wollen uns eine verschachtelte Schleife einmal in einem kurzen Programm anschauen, weil wir dann die Funktion besser verstehen.

```
10 PRINT CHR$(147)
20 FOR I = 1 TO 10
30 FOR J = 1 TO 10
35 IF I*J < 10 THEN PRINT " ";
40 PRINT I*J;
50 NEXT J
60 PRINT
70 NEXT I
```

```
  1   2   3   4   5   6   7   8   9  10

  2   4   6   8  10  12  14  16  18  20

  3   6   9  12  15  18  21  24  27  30

  4   8  12  16  20  24  28  32  36  40

  5  10  15  20  25  30  35  40  45  50

  6  12  18  24  30  36  42  48  54  60

  7  14  21  28  35  42  49  56  63  70

  8  16  24  32  40  48  56  64  72  80

  9  18  27  36  45  54  63  72  81  90

 10  20  30  40  50  60  70  80  90  100

READY.
```

Bild 6-5: Das kleine Einmaleins mit Hilfe verschachtelter Schleifen

Dieses Programm gibt das kleine Einmaleins schön ordentlich auf dem Bildschirm aus. Hätten Sie gedacht, daß das mit einem so kurzen Programm geht? Der Grund liegt in den beiden verschachtelten Schleifen. Die äußere Schleife beginnt bei 20 und endet bei 70. Sie wird 10 mal durchlaufen. Die innere Schleife von 30 bis 60 läuft jeweils ebenfalls 10 mal, also wird sie insgesamt 10 * 10 mal durchlaufen. Wenn Ihnen nicht ganz klar ist, warum wir die Abfrage in Zeile 35 benutzen, dann lassen Sie diese doch einfach einmal weg.

Sie sehen, daß dadurch ein ganz schönes Durcheinander auf dem Bildschirm entsteht, weil die Zahlen nicht gleich lang sind. Durch die Zeile 35 werden auch einstellige Ergebnisse mit zwei Stellen ausgegeben, und die beiden Leerzeichen zwischen den

Ergebnissen setzt der C64 von alleine - eins ist die freie Stelle für das Minuszeichen bei negativen Zahlen.

Zurück zur Balkengruppe

Nach diesem Ausflug in die Welt der verschachtelten Schleifen haben Sie mit dem folgenden Programm wohl kaum noch Schwierigkeiten.

```
10 PRINT CHR$(147)
15 X$ = "****"
20 FOR I = 3 TO 35 STEP 5
30 H = 24 - (24 * RND(1) )
35 FOR K = 24 TO H STEP -1
40 X = I: Y = K
50 GOSUB 10000
60 NEXT K
70 NEXT I
80 GET A$: IF A$ = "" THEN 80
90 PRINT CHR$(147)
100 END
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

```
                              ***
                              ***
                              ***
     ***                      ***  ***
     ***                      ***  ***
     ***                      ***  ***
     ***                      ***  ***
     ***  ***                 ***  ***
     ***  ***                 ***  ***
     ***  ***                 ***  ***
     ***  ***                 ***  ***
     ***  ***            ***  ***  ***
***  ***  ***            ***  ***  ***
***  ***  ***            ***  ***  ***
***  ***  ***            ***  ***  ***
***  ***  ***  ***       ***  ***  ***
***  ***  ***  ***       ***  ***  ***
```

Bild 6-6: Eine Gruppe von Balken - mit Hilfe verschachtelter Schleifen

Obwohl dieses Programm eigentlich nichts Neues mehr enthält, nutzt es die bekannten Befehle doch sehr geschickt aus. Deshalb wollen wir einige Programmzeilen etwas genauer erklären. In Zeile 20 beginnt die äußere Schleife, die 7mal durchlaufen wird, auch wenn man es ihr nicht direkt ansieht. Eigenartigerweise arbeitet sie nämlich in Fünfer-Schritten. Der Grund liegt darin, daß wir ja nachher die Balken schön ordentlich nebeneinander setzen wollen. Durch diese Schrittweite für die Laufvariable I können wir sie nachher direkt in X übertragen, um die Spaltenposition festzulegen.

In Zeile 30 wird die Höhe für jeden Balken berechnet. Dabei entstehen nur Werte zwischen 0 und 23. Da wir aber von unten nach oben zeichnen wollen, interressiert uns die Position von unten gerechnet. Deshalb ziehen wir den erhaltenen Wert von 24 (der tiefsten Position) ab. Vielleicht macht ein Beispiel diese etwas komplizierte Rechnung klarer. Angenommen, die Formel (22 * RND(1)) liefert den Wert 5. Dann soll der Balken also aus 5

Zeilen-Schichten bestehen. Dann müssen wir von 24 (ganz unten) bis 24-5=19 unsere Zeilen-Schichten erstellen.

Ab Zeile 35 beginnt dann die innere Schleife, die von hohen Werten zu kleineren Werten läuft und daher ein STEP -1 benötigt. Die Laufvariable darf nicht wie gewohnt J heißen, da J ja in dem Unterprogramm benutzt wird. Die Schleifenvariable K gibt für den folgenden Unterprogrammaufruf gleich die richtige Zeile, die vorher natürlich in die Variable Y übertragen werden muß. I enthält für jeden kompletten Balken eine um 5 nach rechts versetzte Spaltenposition.

Es ist schon beeindruckend, wie dieses kurze Programm die Balken auf den Bildschirm zaubert. Nachdem uns nun auch Balken auf dem Bildschirm keine Probleme mehr machen, können wir endlich mit unserem Programm Balkengrafik beginnen.

6.6 Der Anfang für das Programm Balkengrafik

Nun kann es endlich losgehen mit unserem größeren Projekt Balkengrafik. Im Gegensatz zu unseren bisherigen Balkenprogrammen, bei denen wir die Balken einfach auf den leeren Bildschirm geschrieben haben, wollen wir diesmal von vornherein für ein ansprechendes Äußeres sorgen. Dazu wollen wir die Grafik mit sogenannten Achsen versehen. Das sind ein horizontaler und ein vertikaler Strich, die gegebenenfalls noch mit einer Einteilung versehen werden können. Da wir später mit unserer Balkengrafik Ausgaben für jeden Monat darstellen wollen, soll die untere Achse mit den Monatsnamen beschriftet werden.

Achsen für die Balkengrafik mit der ASCII-Grafik

Nun könnten wir natürlich an dieser Stelle den Bindestrich als waagerechte Achse und das Ausrufungszeichen als senkrechten Strich benutzen, aber wir wollen Ihnen lieber endlich zeigen, warum die ASCII-Grafik des C64 interessant und reizvoll ist, denn Sterne machen zwar einen schönen Himmel - aber noch keine gute Grafik. Da hat Ihr C64 noch einiges mehr zu bieten. Wenn Sie einmal von vorne auf die Tastatur schauen, sehen Sie,

daß dort jeweils zwei Kästchen aufgedruckt sind. Diese zeigen die noch verfügbaren Zeichen, die Sie neben den Buchstaben und Zahlen auch noch darstellen können.

Sie können jeweils das linke der beiden Zeichen erreichen, indem Sie die dazugehörige Taste zusammen mit der C=-Taste drücken und das rechte mit Shift und C=-Taste. Probieren Sie doch diese Zeichen einfach aus, aus ihnen lassen sich sehr schöne grafische Darstellungen herstellen. Das funktioniert allerdings nur im Großschrift-Modus, den Sie ja hoffentlich wieder eingeschaltet haben. Andernfalls erscheinen statt der Großbuchstaben Kleinbuchstaben auf dem Bildschirm und statt der rechten Grafikzeichen die Großbuchstaben.

Um nun die ASCII-Werte für diese neuen Grafikzeichen herauszubekommen, können Sie wieder unser kleines Programm mit dem ASC-Befehl benutzen oder im Anhang des Commodore-Handbuches nachschauen. Wir jedenfalls werden die Achsen für unsere Balkengrafik aus diesen Zeichen zusammensetzen. Laden Sie bitte zuerst das Unterprogramm UPZEICHEN von der Diskette, denn dieses Unterprogramm zum Setzen der Punkte benötigen wir bald auch in unserem Balkenprogramm. Dann geben Sie zusätzlich die folgenden Zeilen ein:

```
10 PRINT CHR$(147)
15 PRINT"              MONATSSTATISTIK"
20 FOR I = 1 TO 21
30 PRINT"  ";CHR$(180)
40 NEXT I
50 PRINT"  ";CHR$(108);
60 FOR I = 3 TO 39
70 PRINT CHR$(175);
80 NEXT I
90 PRINT"   JA FE MA AP MA JU JU AU SE OK NO DE";
900 GET A$: IF A$ = "" THEN 900
910 PRINT CHR$(147)
920 END
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
```

```
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

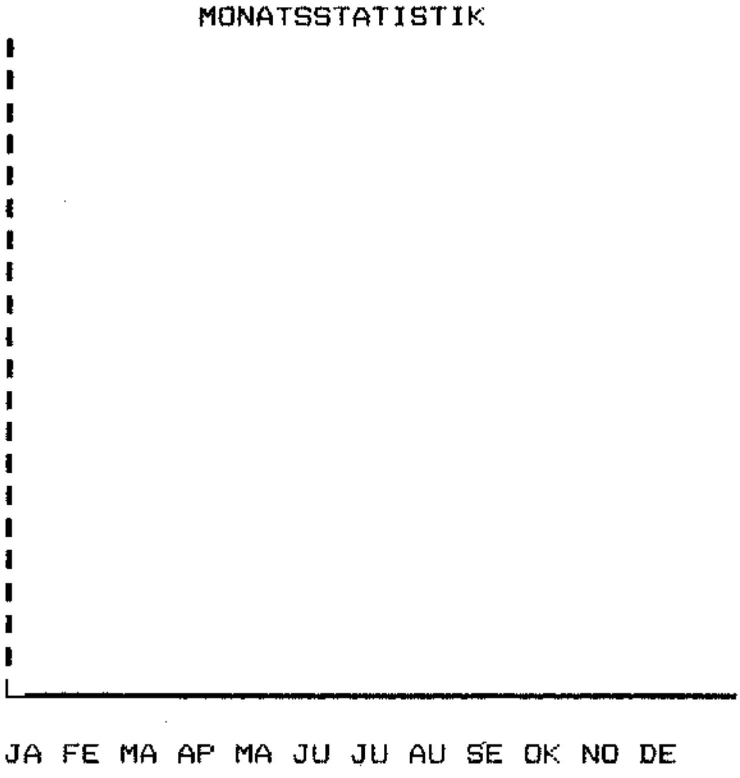


Bild 6-7: Balkengrafik mit Achsen

Wenn Sie das Programm starten, erhalten Sie schon einmal einen ansprechenden Bildschirmaufbau.

Eingabe fester Balkenwerte

Nun wollen wir die Werte für die einzelnen Monate festlegen. Im Augenblick werden wir für jeden Monat einfach eine Variable nehmen, später werden wir sehen, wie man viele Daten in einem Programm einfacher verarbeiten kann. Fügen Sie deshalb erst einmal folgende Zeilen zu Ihrem Programm hinzu:

```
100 JA = 1800
110 FE = 1700
120 MR = 1900
130 AP = 1800
140 MA = 1700
150 JU = 1600
160 JL = 1600
170 AU = 1500
180 SE = 1500
190 OK = 1600
200 NO = 1700
210 DE = 1900
```

Sie können hier auch andere Zahlen verwenden, sollten aber darauf achten, daß diese zwischen 100 und 2000 liegen. Wir werden nämlich nachher bei der Berechnung der Balkenhöhe davon ausgehen, daß die Monatswerte in diesem Bereich liegen, sonst gibt es Schwierigkeiten mit der Darstellung der Balken auf dem Bildschirm.

Mit dem Januar beginnt es

Nun wollen wir einmal versuchen, den Balken für den Januar probeweise zu zeichnen. Dazu benutzen wir im wesentlichen dieselben Zeilen wie im vorherigen Programm. Fügen Sie also noch die Zeilen 500 bis 550 dazu, und dann sollte Ihr Programm folgendermaßen aussehen:

```
10 PRINT CHR$(147)
15 PRINT"              MONATSSTATISTIK"
20 FOR I = 1 TO 21
30 PRINT"  ";CHR$(180)
40 NEXT I
50 PRINT"  ";CHR$(108);
60 FOR I = 3 TO 39
```

```
70 PRINT CHR$(175);
80 NEXT I
90 PRINT"  JA FE MA AP MA JU JU AU SE OK NO DE";
100 JA = 1800
110 FE = 1700
120 MR = 1900
130 AP = 1800
140 MA = 1700
150 JU = 1600
160 JL = 1600
170 AU = 1500
180 SE = 1500
190 OK = 1600
200 NO = 1700
210 DE = 1900
500 BA% = JAN/100
510 BA% = 22 - BA%
520 FOR I = 22 TO BA% STEP -1
530 X = 3: Y = I: X$ = "***"
540 GOSUB 10000
550 NEXT I
900 GET A$: IF A$ = "" THEN 900
910 PRINT CHR$(147)
920 END
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

```
                MONATSSTATISTIK
I
I
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
I **
L____________________________________________

   JA FE MA AP MA JU JU AU SE OK NO DE
```

Bild 6-8: Der erste Monatsbalken ist fertig

Das ist doch schon ein schöner Erfolg, was unser Programm da auf dem Bildschirm produziert. Nun können wir ja daran gehen, die Balken für die übrigen Monate zu erzeugen. Wahrscheinlich werden Sie uns dafür gar nicht mehr benötigen. Aber HALT! Sie müßten dann für jeden Monat wieder neu die Zeilen 500 bis 550 erzeugen. Das ist nicht nur viel Arbeit, sondern führt auch dazu, daß das Programm dadurch sehr lang wird. Woran liegt das, daß wir plötzlich die Schleife zum Erzeugen der Balken für jeden Balken hinschreiben müssen und in dem vorherigen Programm, bei dem die Balkenhöhe per Zufall erzeugt wurde, einfach eine Schleife nehmen konnten.

Nun, im vorherigen Programm konnten wir immer denselben Variablennamen benutzen und daher eine Schleife nehmen. Hier haben wir nun aber 12 verschiedene Variablennamen, und so einfach geht das nicht mehr. Später werden wir eine Reihe von Befehlen kennenlernen, mit denen man in BASIC genau diese Probleme lösen kann. Im Augenblich wollen wir einen einfacheren Weg gehen und die beiden Befehle GOSUB und RETURN einsetzen.

6.7 Das fertige Programm BALKENGRAFIK

Mit diesen Befehlen können wir unser Programm BALKENGRAFIK nun fertigstellen. Den Programmteil, der den eigentlichen Balken zeichnet, legen wir als Unterprogramm an. Dann können wir dies Unterprogramm für jeden Monat einmal aufrufen und übergeben jeweils den Wert für diesen Monat, wie in dem Programm mit der Umrechnung DM in DOLLAR. Damit sieht unser Programm folgendermaßen aus:

```
10 PRINT CHR$(147)
15 PRINT"              MONATSSTATISTIK"
20 FOR I = 1 TO 21
30 PRINT"  ";CHR$(180)
40 NEXT I
50 PRINT"  ";CHR$(108);
60 FOR I = 3 TO 39
70 PRINT CHR$(175);
80 NEXT I
90 PRINT"   JA FE MA AP MA JU JU AU SE OK NO DE";
100 JA = 1800
110 FE = 1700
120 MR = 1900
130 AP = 1800
140 MA = 1700
150 JU = 1600
160 JL = 1600
170 AU = 1500
180 SE = 1500
190 OK = 1600
200 NO = 1700
210 DE = 1900
215 X = 0: X$ = "**"
```

```
220 BA% = JA/100: X=X+3: GOSUB 500
230 BA% = FE/100: X=X+3: GOSUB 500
240 BA% = MR/100: X=X+3: GOSUB 500
250 BA% = AP/100: X=X+3: GOSUB 500
260 BA% = MA/100: X=X+3: GOSUB 500
270 BA% = JU/100: X=X+3: GOSUB 500
280 BA% = JL/100: X=X+3: GOSUB 500
290 BA% = AU/100: X=X+3: GOSUB 500
300 BA% = SE/100: X=X+3: GOSUB 500
310 BA% = OK/100: X=X+3: GOSUB 500
320 BA% = NO/100: X=X+3: GOSUB 500
330 BA% = DE/100: X=X+3: GOSUB 500
340 GET A$: IF A$ = "" THEN 340
350 PRINT CHR$(147)
360 END
500 BA% = 22 - BA%
510 FOR I = 22 TO BA% STEP -1
520 Y = I:
530 GOSUB 10000
540 NEXT I
550 RETURN
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

```
          MONATSSTATISTIK
I
I         **                                      **
I **      **  **                                  **
I **  **  **  **  **                          **  **
I **  **  **  **  **  **  **              **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
I **  **  **  **  **  **  **  **  **  **  **  **
L_______________________________________________________

  JA  FE  MA  AP  MA  JU  JU  AU  SE  OK  NO  DE
```

Bild 6-9: Das fertige Programm "Balkengrafik"

Sie können es ruhig sofort ausprobieren. Anschließend wollen wir dann wieder einige Zeilen des Programms genauer erklären.

1. Die Zeilen 220 - 330 entsprechen den schon bekannten Zeilen im DM-Umrechenprogramm.

2. Die sich ständig wiederholende Erhöhung X = X + 3 verschiebt die Balken nach rechts. Anschließend wird jeweils das Unterprogramm ab Zeile 500 aufgerufen, das die einzelnen Teile eines Balkens zeichnet.

3. Diese fast identischen Zeilen brauchen Sie natürlich nicht jeweils neu zu tippen. Sie können sie - wie Sie sicher noch wissen - einfach dadurch kopieren, daß Sie eine neue Zeilennummer über die alten schreiben, und mit der Eingabe-Taste abschließen. Zum Schluß brauchen Sie nur noch das, was anders sein soll, ändern.

4. Die Tastaturabfrage wurde nach 340-360 verlegt. Auch hier können Sie sich Arbeit ersparen, indem Sie die neuen Zeilennummern über die alten schreiben und die alten Zeilen anschließend löschen.

5. Bei 360 haben wir ein END eingefügt, um das folgende GOSUB-Unterprogramm vom übrigen Programm abzugrenzen.

6. Dieses Unterprogram ab Zeile 500 berechnet aus der Balkenhöhe die einzelnen Abschnitte eines Balkens.

7. Natürlich können Sie in das Programm auch leicht andere Werte einfügen, indem Sie in den Zeilen 100 bis 210 die Werte durch solche ersetzen, die für Sie interessanter sind.

Im übrigen wünschen wir Ihnen mit diesem sehr praktischen Programm viel Spaß.

6.8 Zusammenfassung

1. Unter ASCII-GRAFIK verstehen wir in diesem Buch einfache grafische Darstellungen, bei denen nicht die hochauflösende Grafik des C64 benutzt wird. Vielmehr greifen wir dabei auf die sogenannten ASCII-Zeichen zurück, die beim C64 sofort zur Verfügung stehen und viel einfacher zu benutzen sind. Außerdem nutzen wir ein eigenes Unterprogramm, mit denen man Zeichen auf dem Bildschirm positionieren kann.

2. Der Name ASCII-Zeichen kommt von "American Standard Code for Information Interchange". Darunter versteht man genaue Absprachen darüber, welche Zahl der Computer einem Zeichen zuordnet. Denn unser C64 kann natürlich nur in Zahlen "denken" und muß jedes über die Tastatur

oder sonstwie an ihn übergebene Zeichen erst in eine solche Zahl übersetzen. Die Ziffern beginnen zum Beispiel bei der Nummer 49 (49 = 1), die Buchstaben bei der Nummer 65 (65 = A).

Im Zusammenhang mit dem C64 sind nun bestimmte Zeichen interessant, weil sie keine Buchstaben mehr darstellen, sondern wie kleine Grafikelemente aussehen. Erreicht werden diese Zeichen, indem Sie zusätzlich zu einer Taste die Commodore-Taste oder die Shift-Taste drücken. Wichtig ist dabei, daß sich der C64 im Großschrift-Modus befindet. Dann können Sie die Zeichen vorne auf den Tasten ablesen. Das rechte Zeichen gilt für die Shift-Kombination, das linke Zeichen für die Taste zusammen mit der Commodore-Taste.

Eine Liste aller darstellbarer Zeichen befindet sich im Anhang Ihres C64-Handbuches.

3. Unser Unterprogramm ab Zeile 10000 gibt ein Zeichen an der gewünschten Stelle auf dem Bildschirm aus. Dazu muß vorher die Variable X mit der Spalte, die Variable Y mit der Zeile dieser Stelle versorgt werden. In X$ muß das gewünschte Zeichen abgelegt sein. Hier sind grundsätzlich auch mehrere Zeichen möglich (X$ = "**"), dabei müssen Sie aber am rechten Rand des Bildschirms mit Störungen rechnen.

   Die maximal mögliche Zeile ist 24, die größte Spalte 39. D.h. der linke obere Eckpunkt des Bildschirms wird mit X=0 und Y=0, der untere rechte Eckpunkt des Bildschirms mit X=39 und Y=24 erreicht.

4. Mit dem Befehl GET kann die Tastatur abgefragt werden. Üblich ist der Einsatz in folgender Form:

```
340 GET A$: IF A$ = "" THEN 340
```

   In diesem Fall wartet das Programm, bis eine Taste gedrückt wird. Das zugehörige Zeichen befindet sich anschließend in der Variablen A$. Wurde allerdings schon vorher eine Taste gedrückt, so wird dieses Zeichen genommen und die Zeile sofort beendet.

5. Der Befehl GOSUB 500 löst zunächst ebenso wie GOTO 500 einen Sprung in die Programmzeile 500 aus. Der Unterschied und Vorteil besteht aber darin, daß sich BASIC die Absprungstelle merkt und am Ende des GOSUB-Unterprogramms zu ihr zurückkehrt. Am Ende findet BASIC nämlich den Befehl RETURN, der es zur Rückkehr auffordert. Nach einem GOTO-Sprung würde das Programm einfach an der Zielstelle weiterlaufen. Der besondere Vorteil der Kombination von GOSUB und RETURN besteht gerade darin, daß man häufig benutzte Routinen (etwa eine Umrechnung) in ein GOSUB-Unterprogramm verlegt, wo sie von verschiedenen Stellen des Programms schnell mal eben angesprungen werden können. Diese Technik haben wir beispielsweise eingesetzt, um ein Zeichen an beliebiger Stelle auf dem Bildschirm ausgeben zu können.

7. Daten in Programmen

Im Verlauf der letzten Kapitel haben Sie sicherlich gemerkt, daß ein BASIC-Programm meist nichts anderes tut, als Daten zu bearbeiten. Der besondere Vorteil eines Computers besteht darin, daß er nicht mit festen Zahlen rechnet (wie beispielsweise ein Taschenrechner), sondern Variablen benutzen kann. Inzwischen haben Sie einige Möglichkeiten kennengelernt, wie die Variablen mit den Daten versorgt werden können. Eine besonders einfache Vorgehensweise besteht darin, den Variablen im Programm feste Werte zuzuweisen. Diese Methode haben wir beispielsweise im Programm BENZINVERBRAUCH kennengelernt und auch in der letzten Fassung der Balkengrafik verwendet. Allerdings mußten dort 12 Zeilen (ab Zeile 100) verwendet werden, um die Daten bereitzustellen - ein etwas umständlicher Weg.

Später haben wir ebenfalls in diesem Programm eine Methode vorgestellt, die Daten während des Programmablaufs über die Tastatur zu holen. Dazu dient der Befehl INPUT. Leider müssen bei dieser Form der Dateneingabe die Daten zu jedem neuen Programmablauf auch erneut eingegeben werden. Deshalb ist diese Methode in vielen Fällen nicht geeignet. In diesem Kapitel wollen wir Ihnen nun zwei weitere Möglichkeiten der Datenspeicherung vorstellen, die Sie bald gerne in vielen eigenen Programmen einsetzen werden.

7.1 So einfach kommt ein Programm an seine Daten: READ und DATA

Sicherlich haben Sie das Programm BALKENGRAFIK nun schon mit verschiedenen Monatswerten ausprobiert. Allerdings ist es etwas umständlich, die 12 Werte für die verschiedenen Monate einzutragen, weil dazu 12 Zeilen im Programm geändert werden müssen. Da wir aber versprochen hatten, Ihnen eine elegantere Lösung für dieses Problem zu zeigen, wollen wir Sie nun mit zwei neuen Befehlen bekannt machen: READ und DATA. Sie erinnern sich vielleicht noch daran, daß wir durch die 12 verschiedenen Variablennamen für die Monate Schwierigkeiten

mit unserer Balken-Zeichnen-Routine hatten. Mit den beiden neuen Befehlen können wir wieder einen einzigen Variablennamen verwenden.

Anstatt die 12 Werte in 12 verschiedenen Variablen abzulegen, legen wir sie einfach in Programmzeilen ab, die mit dem Befehl DATA beginnen. Jedesmal, wenn wir nun den Befehl READ benutzen, wird der nächste Wert aus den DATA-Zeilen geholt.

Wie die Befehle READ und DATA funktionieren

Ein ganz einfaches Programm, das dieses Verfahren benutzt, könnte so aussehen:

```
10 READ X
20 PRINT X
30 END
40 DATA 100
```

Dieses Programm ist zwar sehr umständlich, zeigt aber die Benutzung der beiden Befehle. In Zeile 10 liest das Programm die Zahl aus der Zeile 40 und legt sie in X ab. Wollen Sie mehrere Zahlen in einer DATA-Zeile ablegen, so werden die Zahlen durch Komma getrennt. Wir wollen das an einem kleinen Programm verdeutlichen, das einen Einkaufszettel zusammenzählen kann:

```
10 REM Einkaufszettel mit Read und Data
20 G = 0
30 READ A
40 FOR I = 1 TO A
50 READ X
60 G = G + X
70 NEXT I
80 PRINT "Gesamtbetrag: "; G
90 END
100 DATA 5, 13.8, 3.9, 4.77, 39.45, 8.99
```

Dadurch, daß die einzelnen Werte ordentlich in einer Zeile stehen, können Sie bei einem fragwürdigen Ergebnis schnell kontrollieren, ob Sie sich nicht vertippt haben. Wahrscheinlich ist Ihnen eine Besonderheit in unserem Programm aufgefallen. Der

erste Wert in der DATA-Zeile ist gar keine Ausgabe, sondern gibt die Anzahl der vorhandenen Ausgaben in der DATA-Zeile an. Wir könnten natürlich auch oben im Programm A auf den jeweils richtigen Wert setzen, aber dann müßte man nach der Eingabe der Daten erst noch oben die Variable ändern. So kann das in der DATA-Zeile direkt geschehen.

Einbeziehung von READ/DATA ins Programm "Balkengrafik"

Nun wollen wir unser Programm BALKENGRAFIK um die beiden neuen Befehle erweitern und damit gehörig vereinfachen. Damit Sie das erheblich geänderte Programm nicht völlig neu abtippen müssen, wollen wir kurz erklären, wie Sie am einfachsten die neue Programmversion erstellen können. Laden Sie das Programm BALKENGRAFIK, und speichern Sie es sofort wieder unter einem neuen Namen ab, beispielsweise mit SAVE "BALKNEU",8. Dadurch verhindern Sie, daß Sie nach der Änderung beim Abspeichern versehentlich das bisherige Programm überschreiben und damit löschen.

Hinweis: Es gibt übrigens eine komfortablere Möglichkeit, Programme mit langen Namen zu laden. Und BALKENGRAFIK ist auch schon ganz schön lang. Wie Sie sich die Tipperei zumindest erleichtern können, finden Sie im Lexikon unter LOAD!

Was nun das geladene Programm angeht, so müssen Sie im wesentlichen die Zeilen 100 bis 210 und die Zeilen 220 bis 330 löschen. Das geht recht einfach, indem Sie die Zeilennummer 100 in eine leere Zeile schreiben und die Eingabetaste betätigen. Dann ändern Sie die 100 in 110 und drücken wieder die Eingabetaste und so fort für alle Zeilennummern, deren Zeilen Sie löschen wollen.

Nun brauchen Sie nur noch die Zeile 215 zu löschen und die Zeilen 200 bis 260 und 800 - 810 einzugeben. Dann sollte Ihr Programm folgendermaßen aussehen:

```
10 PRINT CHR$(147)
15 PRINT"              MONATSSTATISTIK"
20 FOR I = 1 TO 21
```

```
30 PRINT"  ";CHR$(180)
40 NEXT I
50 PRINT"  ";CHR$(108);
60 FOR I = 3 TO 39
70 PRINT CHR$(175);
80 NEXT I
90 PRINT"   JA FE MA AP MA JU JU AU SE OK NO DE";
200 X = 3: X$ = "***"
210 FOR B = 1 TO 12
220 READ BA%
230 BA% = BA% / 100
240 GOSUB 500
250 X = X+3
260 NEXT B
340 GET A$: IF A$ = "" THEN 340
350 PRINT CHR$(147)
360 END
500 BA% = 22 - BA%
510 FOR I = 22 TO BA% STEP -1
520 Y = I:
530 GOSUB 10000
540 NEXT I
550 RETURN
800 DATA 1900,1800,1600,1500,1300,1000
810 DATA 1200,1200,1700,1800,1900,1900
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

Nun einiges zur Erklärung: In Zeile 200 setzen wir den Startwert für die X-Positon auf 3. Dadurch wird der erste Balken ab der 3. Spalte gezeichnet. Von Zeile 210 bis 260 geht unsere Schleife, die nacheinander die Werte aus den beiden DATA-Zeilen einliest, durch 100 teilt und das Unterprogramm ab Zeile 500 aufruft. Für die Schleifenvariable durften wir übrigens nicht das gewohnte I oder J nehmen, da beide Variablen durch die Unter-

programme verändert werden. Anschließend wird noch die X-Position um 3 erhöht.

Wichtiger Hinweis: Achten Sie bei der Benutzung der Befehle READ und DATA bitte darauf, daß für jeden READ-Befehl auch ein DATA-Wert vorhanden ist. Trennen Sie alle Werte durch Komma, und verwechseln Sie das Komma bitte nicht mit dem Dezimalpunkt und umgekehrt. Hier können recht leicht Fehler (Fehlermeldung: OUT OF DATA) entstehen, die nur schwer zu finden und zu beheben sind.

7.2 Felder - viele Daten unter einem Namen

Bei unserem Programm BALKENGRAFIK waren wir auf die Schwierigkeit gestoßen, mehrere Werte in Variablen ablegen zu müssen. Wir haben dazu genauso viele Variablennamen benutzt, konnten aber nun die Werte nicht mehr so einfach in einer Schleife weiterverarbeiten. In diesem Abschnitt wollen wir Ihnen zeigen, wie man viele Daten unter einem Namen ablegen kann und dann auf diese in einer Schleife zugreift. Das Zauberwort lautet: Felder. Um es gleich vorwegzunehmen: Vielen Programmierern machen Felder erhebliche Schwierigkeiten, und wir haben lange überlegt, ob dieses Thema überhaupt in ein Einsteigerbuch gehört. Aber vielleicht können wir ja gerade durch dieses Kapitel Ihre Anfangsschwierigkeiten mit Feldern möglichst gering halten.

Was sind Felder und wie funktionieren sie?

Nun wollen wir aber erst einmal erklären, was sich hinter einem Feld verbirgt. Sie erinnern sich doch sicherlich noch an die erste Version des Programms BALKENGRAFIK, die folgende Zeilen enthielt:

```
100 JA = 1800
110 FE = 1700
120 MR = 1900
130 AP = 1800
140 MA = 1700
```

```
150 JU = 1600
160 JL = 1600
170 AU = 1500
180 SE = 1500
190 OK = 1600
200 NO = 1700
210 DE = 1900
```

Es wäre doch eigentlich viel angenehmer, wenn man die 12 verschiedenen Werte in einer einzigen Variablen ablegen könnte. Und wenn man dann den Wert für den dritten Monat benötigt (MR), sagt man BASIC nur noch: Ich benötige den dritten Wert aus der Variablen. Dadurch, daß man zur Bestimmung des Monats wieder eine Zahl verwenden kann (hier: 3), werden auch wieder Schleifen möglich. Eine Schleife für alle 12 Monate könnte dann etwa folgendermaßen aussehen (nur symbolisch, nicht in echter BASIC-Befehlssyntax geschrieben):

```
FOR I = 1 TO 12
Gib mir den i.ten Monatswert
Tu etwas mit dem Wert
NEXT I
```

Um diese Schreibweise in ein funktionsfähiges Programm umzuändern, müssen wir uns natürlich erst einmal einen Namen für diese neue Variable, die alle 12 Werte enthalten soll, einfallen lassen. Wir wollen sie sinnvollerweise MO (für: MONAT) nennen. Als nächstes müssen wir BASIC mitteilen, daß MO keine normale Variable ist (Integervariable, Stringvariable usw.), sondern ein Feld. Dazu will BASIC von uns aber wissen, wie viele Werte wir denn in unserer neuen Variablen abzulegen gedenken. Dafür lautet der Befehl:

```
DIM MO(12)
```

Taucht dieser Befehl in einem Programm zu Anfang auf, so weiß der C64 ab dann, daß die neue Variable MO ein Feld ist und 12 verschiedene Werte enthalten soll. Wenn wir später im Programm den dritten Wert aus der Variablen (dem Feld) benötigen, so schreiben wir einfach MO(3), und BASIC holt uns den richtigen Wert. Wollen wir beispielsweise den dritten Wert auf 1800 setzen, so schreiben wir:

```
MO(3) = 1800
```

Das Schwierige bei der Arbeit mit einem Feld ist, daß komplizierter gedacht werden muß: Man muß nämlich jeweils überlegen, welcher Monat der März ist und diese Zahl dann im Befehl in die Klammer schreiben. Wollen wir beispielsweise die Monate von Juni bis Dezember auf dem Bildschirm ausgeben, so müssen wir überlegen, welche Nummer der Juni hat und welche der Dezember und können dann erst schreiben:

```
FOR I = 6 TO 12
PRINT MO(I)
NEXT I
```

So können diese drei Zeilen natürlich noch nicht verwendet werden, weil wir ja noch gar keine Werte in der Variablen für die Monate abgelegt haben. Aber es wird schon deutlich, wie wir plötzlich wieder in einer Schleife auf mehrere Variablen (oder besser auf die Werte in einem Feld) zugreifen können.

Die erste praktische Anwendung eines Feldes

Nun wollen wir aber endlich mit einem kleinen Programm zeigen, wie die Arbeit mit einem Feld im Programm richtig aussieht. Das folgende Programm legt ein Feld für 12 Werte an, liest diese Werte aus DATA-Zeilen und schreibt anschließend beliebige Werte des Feldes auf den Bildschirm:

```
10 PRINT CHR$(147)
20 DIM MO(12)
30 FOR I = 1 TO 12
40 READ MO(I)
50 PRINT MO(I)
60 NEXT I
70 PRINT "Bitte eine Zahl zwischen 1 und 12 eingeben"
80 PRINT "Eine 0 beendet das Programm"
90 INPUT X
100 IF X = 0 THEN END
110 IF X < 0 OR X > 12 THEN GOTO 70
120 PRINT "Der "; X;".te Monat hat den Wert: ";MO(X)
130 GOTO 70
200 DATA  500, 900,1300,1500,1400,1600
210 DATA 1800,1900,1700,1100, 400, 800
```

Zur Erklärung: Zuerst teilen wir dem C64 in Zeile 20 mit, daß MO ein Feld mit 12 möglichen Werten ist. In der folgenden Schleife (30 - 60) lesen wir die 12 Werte aus den beiden DATA-Zeilen und geben sie auch gleich auf dem Bildschirm aus. In Zeile 70 bis 90 holen wir eine Zahl zwischen 1 und 12 in die Variable X. Bei einer 0 beenden wir, wie versprochen, das Programm, ansonsten prüfen wir, ob die Zahl X im erlaubten Bereich liegt. Ist das nicht der Fall, springen wir zur erneuten Eingabe einer Zahl, weil BASIC sonst eine Fehlermeldung erzeugen würde. - Wir müssen uns schon strikt an die Vereinbarung halten, daß MO nur 12 Werte enthalten kann.

In Zeile 120 geben wir dann den Wert für den Monat aus. Der entscheidende Teil dieser Zeile lautet: MO(X). Der in dieser Zeile ausgegebene Text ist geschickt aus der Zahl X und ".te" zusammengesetzt. Das Ergebnis sieht dann auf dem Bildschirm besser aus. In Zeile 130 geht es dann wieder zur Eingabe einer Zahl. Am Schluß des Programms stehen die beiden DATA-Zeilen mit den 12 Werten für die Monate.

Vielleicht beginnen Sie jetzt, den Wert und den Nutzen von Feldern zu schätzen. Hätten wir nämlich kein Feld benutzt, sondern die 12 Werte in den 12 Variablen JA bis DE abgelegt, so könnten wir die Augabe eines beliebigen Monatswertes nur durch eine lange Abfrage erreichen, etwa in der Art:

```
IF X = 1 THEN PRINT"Der "; X;".te Monat hat den Wert: ";JA
IF X = 2 THEN PRINT"Der "; X;".te Monat hat den Wert: ";FE
IF X = 3 THEN PRINT"Der "; X;".te Monat hat den Wert: ";MR
```

und so weiter.

7.3 BALKENGRAFIK mit Feldern

Mit diesem Wissen um Felder können wir nun auch unserem Programm BALKENGRAFIK den letzten Schliff geben. Wir lesen die Werte für die einzelnen Monate aus DATA-Zeilen und legen sie in einem Feld ab. An der Funktion des Programms ändert sich dadurch zwar nichts, aber Sie sehen noch einmal den

richtigen Einsatz der neuen Befehle in einem Programm und können dieses für eigene Zwecke viel leichter erweitern.

```
5 DIM MO(12)
10 PRINT CHR$(147)
15 PRINT"                    MONATSSTATISTIK"
20 FOR I = 1 TO 21
30 PRINT"  ";CHR$(180)
40 NEXT I
50 PRINT"  ";CHR$(108);
60 FOR I = 3 TO 39
70 PRINT CHR$(175);
80 NEXT I
90 PRINT"   JA FE MA AP MA JU JU AU SE OK NO DE";
200 X = 3: X$ = "**"
210 FOR B = 1 TO 12
220 READ MO(B)
230 BA% = MO(B) / 100
240 GOSUB 500
250 X = X+3
260 NEXT B
340 GET A$: IF A$ = "" THEN 340
350 PRINT CHR$(147)
360 END
500 BA% = 22 - BA%
510 FOR I = 22 TO BA% STEP -1
520 Y = I:
530 GOSUB 10000
540 NEXT I
550 RETURN
800 DATA 1900,1800,1600,1500,1300,1000
810 DATA 1200,1200,1700,1800,1900,1900
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

```
          MONATSSTATISTIK
       DURCHSCHNITT 1566.66667
I **                             ** **
I ** **                       ** ** **
I ** **                    ** ** ** **
I ** ** **                 ** ** ** **
I ** ** ** **              ** ** ** **
I ** ** ** **              ** ** ** **
I ** ** ** ** **           ** ** ** **
I ** ** ** ** **    ** **  ** ** ** **
I ** ** ** ** **    ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
I ** ** ** ** ** ** ** **  ** ** ** **
L______________________________________

  JA FE MA AP MA JU JU AU SE OK NO DE
```

Bild 7-1: **Die Balkengrafik mit Angabe des Durchschnittswertes**

Wir hoffen, daß Sie mit diesem Programm nicht nur viel Spaß haben, sondern daß es Sie vielleicht auch dazu anregt, einiges weiter auszuprobieren und Veränderungen vorzunehmen. So können Sie das Programm zum Beispiel leicht um einige Zeilen erweitern, die den Durchschnitt für alle 12 Monate berechnen. Diese Zeilen könnten beispielsweise lauten:

```
280 S = 0
290 FOR I = 1 TO 12
300 S = S + MO(I)
310 NEXT I
```

```
320 D = S / 12
330 PRINT CHR$(19): PRINT
335 PRINT "        Durchschnitt : "; D
```

Sicherlich fallen Ihnen noch viele weitere Möglichkeiten für Änderungen ein. Wir wollen es beim Hinweis belassen, daß das Programm natürlich auch 31 Balken für die Darstellung, bezogen auf einzelne Tage, zeichnen kann. Sie müßten dann natürlich das Feld in DIM TA(31) umändern und darauf achten, daß ein Monat ja unterschiedlich viele Tage hat und .... Aber das wollen Sie ja bestimmt selber probieren. Wir werden lieber im diesem Kapitel eine weitere interessante Möglichkeit der Datenspeicherung vorstellen - die sequentiellen Dateien.

7.4 Zwischen-Zusammenfassung

1. Mit dem Befehl READ können Sie Daten aus speziellen Programmzeilen, den sogenannten DATA-Zeilen, einlesen und einer Variablen zuweisen.

```
10 READ B
20 PRINT B
30 DATA 100
```

Der Vorteil liegt unter anderem darin, daß die DATA-Zeilen nicht direkt hinter dem READ-Befehl stehen müssen. Man packt sie meistens komplett, d.h. für alle Variablen, an den Schluß des Programms. Wichtig ist nur, daß die READ-Befehle in der richtigen Reihenfolge auf die DATA-Angaben zugreifen. Diese werden nämlich einfach aneinandergereiht, lediglich durch Komma getrennt:

```
100 DATA 100,220,34.5
```

Daran wird deutlich, daß man bei Werten in DATA-Zeilen besonders darauf achten muß, Punkt und Komma nicht zu verwechseln. Denn nach jedem Komma beginnt für BASIC ein neuer DATA-Wert. Und hier wäre es verhängnisvoll, wenn man aus Versehen die halbe Mark als ,5 eingetragen hätte statt als .5!

Übrigens: Die DATA-Zeilen dürfen durch nichts unterbrochen werden, auch nicht durch REM-Bemerkungen!

2. Mehrere aufeinanderfolgende Programmzeilen können komfortabel gelöscht werden, indem man die Nummer der ersten Zeile eingibt, die Return-Taste drückt und dann jeweils nur die entsprechende Ziffer ändert und durch Return dann auch diese Zeile löscht und so fort.

3. Unter einem Feld oder Array versteht man Variablen, die sich aus mehreren variablen Bestandteilen, also Einzelvariablen zusammensetzen. Nehmen wir eine Variable MO$, und dimensionieren wir sie mit 12. Dies geschieht durch die DIM-Anweisung: DIM MO$ (12).

```
10 DIM MO$(12)
20 FOR I = 1 TO 12
30 READ MO$(I)
40 NEXT I
50 DATA Januar, Februar, März, April, Mai, Juni, Juli, August,
September, Oktober, November, Dezember
```

Am Ende versteht das Programm unter MO$(3) immer den März und unter MO$(10) immer den Oktober.

Solche Felder können auch mehrdimensional sein. Ein zweidimensionales Feld könnte man sich in diesem Fall so vorstellen, daß in senkrechten Zeilen die Monate und in waagerechten Spalten verschiedene Jahre eingetragen werden. Für die Jahre 1987, 1988, 1989 sähe das so aus:

```
10 DIM MO$(12,3)
```

Hier wird festgelegt, daß die Variable über 12 Monatsnamen und drei Jahresnamen verfügen soll. Anschließend kann man die Daten so aufbauen, daß MO$(11,2) der November des zweiten Jahres, also in diesem FAlle: der November 1988 ist usw.

Felder können über noch mehr Dimensionen verfügen, die Vorstellung oder auch grafische Darstellung ist dann aber natürlich nicht mehr so einfach.

7.5 Daten auf Diskette - sequentielle Dateien

In diesem Abschnitt wollen wir Sie nun mit einer besonders leistungsfähigen Form der Dateispeicherung vertraut machen - der Speicherung auf Diskette. Der große Vorteil besteht darin, daß Daten und eigentliches Programm getrennt sind: Einmal ist auf der Diskette beispielsweise unter BALKENGRAFIK das Programm gespeichert, einmal die Gruppe der DATA-Zeilen unter "AUSGABEN88". Nun ist es ein leichtes, die Ausgaben für 1989 unter "AUSGABEN89" ebenfalls auf Diskette zu speichern und mit demselben Programm BALKENGRAFIK anzuzeigen. Daten auf der Diskette bezeichnet man übrigens in der Computersprache als Datei. Auch ein gespeichertes BASIC-Programm ist eine Datei. Sequentielle Dateien sind nun einfach eine besondere Form dieser Dateien. Was den Begriff übrigens angeht: Sequentielle Dateien heißen deshalb so, weil die Daten wie in DATA-Zeilen streng nacheinander angeordnet werden (sequentiell = aufeinander folgend). Bleibt nur die Frage: Wie bekommen wir unsere Daten auf die Diskette und von dort wieder in unser Programm?

Wie arbeitet man mit einer sequentiellen Datei?

Vielleicht erinnern Sie sich noch an den Anfang dieses Buches, wo wir gemeinsam Disketten vorbereitet (formatiert) haben. Dort haben wir eine Verbindung zur Befehlszentrale des Diskettenlaufwerks hergestellt (OPEN 1,8,15) und nach der Übermittlung des Befehls wieder geschlossen (CLOSE 1).

Grundsätzlich ähnlich sind wir verfahren, wenn wir ein Programmlisting auf dem Drucker ausgeben wollten:

```
OPEN 1,4
CMD 1
LIST
PRINT#1
CLOSE 1
```

Nun: So, wie wir Daten zum Drucker schicken können (dem Drucker ist es egal, ob es sich um ein Programmlisting handelt oder Ihre Ausgaben für 1988), so können wir auch Daten zum

Diskettenlaufwerk schicken und damit speichern. Im Gegensatz zum Formatieren sprechen wir dann aber nicht mehr die Befehlszentrale an (die 15 in OPEN 1,8,15), sondern eine der Datenverarbeitungsstellen. Im Gegensatz zum Drucker, der unsere Daten einfach brav an den Druckkopf schickt, braucht die Datenverarbeitungsstelle der Floppy allerdings einen Namen für unsere Daten, damit diese später unter diesem Namen auch wiedergefunden werden können. Zusätzlich muß die Floppy wissen, daß es sich um eine sequentielle Datei handeln soll und daß die Daten geschrieben werden. Übrigens wollen wir im folgenden bei sequentiellen Dateien statt der Zahl 1 im OPEN-Befehl die Zahl 5 verwenden. Dann wissen wir immer sofort Bescheid. Damit sieht das Speichern von Daten auf der Diskette schon mal vereinfacht folgendermaßen aus:

```
10 OPEN 5,8,5,"MEINEDATEN,S,W"
Schicke die Daten zum Diskettenlaufwerk
30 CLOSE 5
```

Die erste "5" vor der "8" stellt die Kennziffer unserer Verbindung her. Da auch mehrere Verbindungen gleichzeitig bestehen können, müssen diese unterscheidbar sein, und das geschieht an Hand der Kennziffer. Damit der Computer weiß, welches Gerät (Floppy oder Drucker) die Daten bekommen soll, hat jedes Gerät eine Nummer, die Geräteadresse. Diese lautet bei der Floppy 8. Auf die Bedeutung der letzten Ziffer wollen wir an dieser Stelle nicht weiter eingehen. Wenn Sie ein Profi der Datenspeicherung werden wollen, sollten Sie mal in "Das große Floppybuch zur 1541" schauen, dort werden selbst die letzten Geheimnisse der Floppy gelüftet. Wenn Sie sich aber einfach an unsere 3 Zahlen halten, kann eigentlich nichts schiefgehen.

In den Anführungsstrichen geben wir durch Komma getrennt den Namen für unsere Datei, ein S für "sequentiell" und ein W für "WRITE" (englisch: Schreiben) an. Nun kann die Datenübertragung beginnen - wir werden gleich sehen, wie das geht - und am Schluß beenden wir unsere Datenspeicherung durch CLOSE 5. Wichtig ist noch, daß nicht schon etwa eine Datei unter diesem Namen auf der Diskette vorhanden ist, das mag die Floppy gar nicht und meldet sich mit einer blinkenden

roten Lampe. In unseren Programmen werden wir diese Möglichkeit geschickt verhindern.

Das Lesen der Daten geschieht dann ganz ähnlich, indem wieder eine Verbindung zur Floppy erstellt wird, mit demselben Namen, aber einem "R" für "READ" (= Lesen), dann können die Daten geholt werden, und am Schluß wird die Verbindung wieder beendet. Grundsätzlich sieht das dann so aus:

```
10 OPEN 5,8,5,"MEINEDATEN,S,R"
Hole die Daten vom Diskettenlaufwerk
30 CLOSE 5
```

Praktisches Beispiel für die Nutzung von sequentiellen Dateien

Nun wollen wir Ihnen aber schnell ein erstes Programm vorstellen, das 5 Daten auf der Diskette ablegt und anschließend wieder holt.

```
10 OPEN 5,8,15,"S:MEINEDATEN":CLOSE 5
20 OPEN 5,8,5,"MEINEDATEN,S,W"
30 FOR I = 1 TO 5
40 READ X
50 PRINT#5,X
60 NEXT I
70 CLOSE 5
80 PRINT "DIE DATEN SIND AUF DISKETTE"
100 OPEN 5,8,5,"S:MEINEDATEN,S,R"
110 FOR I = 1 TO 5
120 INPUT#5,X
130 PRINT X
140 NEXT I
150 CLOSE 5
160 END
170 DATA 1100,1400,1900,1600,900
```

Wenn Sie dieses Programm starten, werden 5 Zahlen in die Datei MEINEDATEN auf der Diskette geschrieben und anschließend wieder gelesen. Diese Datei finden Sie auch im Inhaltsverzeichnis der Diskette, aber ACHTUNG: Speichern Sie erst Ihr Programm auf Diskette, denn durch das Einladen des Inhaltsverzeichnisses wird es gelöscht. Die Daten unterscheiden sich im Inhaltsverzeichnis durch die geänderten drei Buchstaben von an-

deren Dateien: Programme erhalten ein "PRG", sequentielle Dateien ein "SEQ". Nun wollen wir aber noch auf die Funktion des Programms eingehen:

Die Zeile 10 löscht eine möglicherweise auf der Diskette vorhandene Datei MEINEDATEN. Ist noch keine Datei vorhanden, passiert weiter nichts. Zeile 20 stellt dann in schon bekannter Weise eine Verbindung zur Speicherung von Daten her. Man sagt in der Fachsprache auch: Die Datei MEINEDATEN wird auf der Diskette zum Schreiben geöffnet. Anschließend werden in einer Schleife (30 - 60) 5 Zahlen aus der DATA-Zeile am Programmende gelesen. Das Schreiben auf Diskette erfolgt in Zeile 50. Im Gegensatz zur Ausgabe auf den Bildschirm erhält der PRINT-Befehl hier noch ein "#" und die Nummer der Verbindung (hier die 5).

Achten Sie bitte unbedingt darauf, daß Sie das Zeichen "#" direkt, also ohne trennendes Leerzeichen, hinter das "PRINT" schreiben. An dieser Stelle nimmt Ihnen der C64 ausnahmsweise einmal Leerzeichen übel und verwechselt dann PRINT und PRINT#5. Nachdem alle 5 Zahlen auf der Diskette sind, wird die Datei ordnungsgemäß geschlossen. Dies ist wichtig, und sollte eins Ihrer Programme einmal mit einer Fehlermeldung beendet werden, bevor die Zeile CLOSE 5 abgearbeitet wurde, so geben Sie bitte CLOSE 5 im Direktmodus ein. Eine nicht ordnungsgemäß geschlossene Datei erhält im Inhaltsverzeichnis noch ein Sternchen vor den drei Buchstaben, also "*SEQ", und sollte von Ihnen gelöscht werden.

In Zeile 100 wird dann diese Datei zum Lesen geöffnet. Der zugehörige Befehl zum Holen der Daten heißt INPUT#5 und unterscheidet sich vom üblichen INPUT-Befehl wieder durch das "#" und die Verbindungsnummer. Auch hier wird die Datei nach dem Lesen der Zahlen wieder geschlossen (CLOSE 5).

7.6 Was bei der Datenspeicherung auf Diskette zu beachten ist

Wir können Ihnen natürlich in diesem Einsteigerbuch nicht alle Tips und Kniffe der Datenspeicherung verraten, dazu bedarf es schon eines "Großen Floppybuches". Trotzdem möchten wir Ihnen ein paar wichtige Faustregeln und Kniffe verraten, damit Sie nicht wie wir wochenlang herumprobieren müssen, bis die Datenspeicherung sicher klappt.

1. Beim Öffnen der Datei (OPEN) zum Schreiben (W) darf diese Datei noch nicht vorhanden sein, sonst verweigert die Floppy ihre Mitarbeit und zeigt Ihnen die "rote Karte". Wenn die rote Lampe der Floppy blinkt, brauchen Sie keinen Schaden zu befürchten. Das Blinken hört auf, sobald Sie die Floppy ein weiteres Mal (fehlerfrei) benutzen, also beispielsweise ein Programm laden oder speichern. Löschen Sie sicherheitshalber vorher die Datei auf der Diskette, wie es unser Programm in Zeile 10 tut.

2. Wollen Sie eine Datei zum Lesen öffnen und ist diese nicht vorhanden, so blinkt anschließend ebenfalls die Floppy, und es können keine Daten gelesen werden. Dabei kann sich der C64 sogar "aufhängen" (d.h. nichts läuft mehr, der C64 reagiert nicht mehr auf Eingaben, auch nicht auf RUN/STOP), weil er Daten von der Floppy holen will, diese aber keine schicken kann. Drücken Sie RUN/STOP + RESTORE, und beheben Sie anschließend den Fehler im Programm. Das Blinken der Floppy hört nach dem nächsten fehlerfreien Benutzen der Floppy wieder auf, ist also nicht weiter schlimm. Sollte auch RUN/STOP + RESTORE nicht helfen, können Sie den C64 nur aus- und wieder einschalten.

3. Sie können Zahlen und Zeichenketten (Strings) auf der Diskette speichern. Das funktioniert genauso wie bei den Befehlen PRINT und INPUT:

   PRINT#5,X (schreibt die Zahl X auf Diskette)
   INPUT#5,X (holt die Zahl X von Diskette)
   PRINT#5,X$ (schreibt den String X$ auf Diskette)
   INPUT#5,X$ (holt den String X$ von Diskette)

Allerdings dürfen Variablen nicht verwechselt werden. Haben Sie also einen String auf Diskette geschrieben und wollen mit INPUT#5,X eine Zahl holen, dann gibt es erhebliche Schwierigkeiten: Der C64 meldet ein "? FILE DATA ERROR".

4. Sie können zwar mehr Daten auf Diskette schreiben als lesen, aber nicht umgekehrt. Wenn Sie also ein Programm schreiben, das unterschiedlich viele Daten in einer Datei ablegen kann, so sollte dieses Programm zu Beginn die Anzahl der Daten in die Datei schreiben und dann erst die Daten. Beim Lesen kann das Programm dann die Anzahl zuerst holen und anschließend genau die richtige Menge Daten. Unser Beispielprogramm sähe dann folgendermaßen aus:

```
10 OPEN 5,8,15,"S:MEINEDATEN":CLOSE 5
20 OPEN 5,8,5,"MEINEDATEN,S,W"
25 READ A   : REM ANZAHL AUS DATENZEILEN HOLEN
28 PRINT#5,A: REM ANZAHL ZUERST IN DATEI SCHREIBEN
30 FOR I = 1 TO A
40 READ X
50 PRINT#5,X
60 NEXT I
70 CLOSE 5
80 PRINT "DIE DATEN SIND AUF DISKETTE"
100 OPEN 5,8,5,"S:MEINEDATEN,S,R"
105 INPUT#5,A: REM ANZAHL DATEN AUS DER DATEI HOLEN
110 FOR I = 1 TO A
120 INPUT#5,X
130 PRINT X
140 NEXT I
150 CLOSE 5
160 END
170 DATA 5, 1100,1400,1900,1600,900
```

7.7 So kommen die Daten für die Balken auf Diskette

Da wir ja im folgenden das Programm BALKENGRAFIK so abändern wollen, daß es sich die Daten für die Balken von der Diskette holt, müssen die Daten dort natürlich vorhanden sein. Dazu wollen wir ein kleines Programm schreiben, mit dem Sie

diese Daten erstellen und auf Diskette speichern können. Zusätzlich soll das Programm auch Daten von der Diskette lesen können, die Sie dann geändert wieder abspeichern können. Da das Programm die Daten für die Balkenhöhe in einem Feld ablegt, werden Sie durch dieses Beispiel auch etwas sicherer im Umgang mit Feldern.

```
10 REM BALKENDATEN AUF DISKETTE
20 PRINT CHR$(147)
30 DIM BA(12)
40 INPUT "DATEN VON DISKETTE LESEN J/N";N$
50 IF N$ = "J" THEN GOTO 100
60 IF N$ <> "N" THEN GOTO 40
65 REM EINGABE DER DATEN
70 FOR I = 1 TO 12
75 PRINT I;
80 INPUT ".TEN BALKEN "; BA(I)
90 NEXT I
95 GOTO 500 : REM AENDERN
100 REM EINLESEN VON DISKETTE
110 INPUT "BITTE DATEINAMEN EINGEBEN";N$
120 OPEN 5,8,5,N$+",S,R"
130 FOR I = 1 TO 12
140 INPUT#5,BA(I)
150 NEXT I
160 CLOSE 5
170 GOTO 500
500 REM AENDERN DER DATEN
510 PRINT CHR$(147)
520 FOR I = 1 TO 12
530 PRINT I,BA(I)
540 NEXT I
550 PRINT
555 INPUT "NR DES BALKENS 0=ENDE";N
560 IF N = 0 THEN GOTO 800: REM SPEICHERN
565 IF N<0 OR N>12 THEN GOTO 555: REM FALSCHE EINGABE
570 PRINT N,BA(N)
580 INPUT "NEUER WERT";BA(N)
590 GOTO 500 : REM AENDERN
800 INPUT "DATEN SPEICHERN (J)";N$
810 IF N$ <> "J" THEN END : REM NA GUT, WENN ER NICHT WILL !
820 INPUT "BITTE DATEINAMEN EINGEBEN";N$
830 OPEN 5,8,15,"S:"+N$: CLOSE 5
840 OPEN 5,8,5,N$+",S,W"
850 FOR I = 1 TO 12
```

```
860 PRINT#5,BA(I)
870 NEXT I
880 CLOSE 5
890 PRINT "DATEN IN "; N$; " GESPEICHERT."
900 END
```

Speichern Sie das Programm bitte unter dem Namen BALKENEDITOR auf Diskette ab. In Zeile 30 wird das Feld zur Aufnahme der Balkendaten dimensioniert. Anschließend wird gefragt, ob die Daten von der Diskette geholt (J) oder eingegeben (N) werden sollen. Durch die Zeile 60 wird bei einer falschen Eingabe erneut in Zeile 40 gesprungen. Diese Technik der Überprüfung einer Eingabe sollten Sie möglichst auch in eigenen Programmen anwenden. Dazu sollten Sie in der Eingabezeile (hier 40) vorgeben, welche Eingaben erlaubt sind (J/N) und bei einer nicht erlaubten Eingabe eine neue anfordern.

Ab Zeile 65 werden dann in einer Schleife 12 Werte in das Feld geholt. Dabei wird die Nummer des Wertes ausgegeben, indem Sie mit dem Hinweistext geschickt zu einer Zeile wie beispielsweise

```
5 .TEN BALKEN ?
```

zusammengesetzt wird. Nach Eingabe der 12 Werte wird zum Programmteil "Daten ändern" verzweigt.

Ab Zeile 100 werden die Daten von der Diskette geholt. Dazu muß natürlich erst einmal der Dateiname eingegeben werden, denn nur so können die Daten auf der Diskette auch gefunden werden. Wenn Sie hier einen Namen eingeben, zu dem keine Datei auf der Diskette gehört, wird ab Zeile 120 die Floppy blinken, und das Programm wird anschließend 12 Nullen auf dem Bildschirm anzeigen. Merken Sie sich also bitte die Namen der Dateien auf der Diskette. Ist die Datei aber vorhanden, so kann sie in Zeile 120 geöffnet werden. Wenn Ihnen die Funktion der Zeile nicht klar ist, so tragen Sie doch dort einmal für N$ in Gedanken MEINEDATEN ein. Dann steht dort:

```
120 OPEN 5,8,5,"MEINEDATEN"+",S,R"
```

und da ein "+" zwischen Zeichenketten diese einfach zu einer großen zusammensetzt, steht dort nichts anderes als:

```
120 OPEN 5,8,5,"MEINEDATEN,S,R"
```

Genauso werden wir später die richtige Befehlszeile zum Löschen der Datei auf Diskette und zum Schreiben zusammensetzen. Die Schleife von 130-150 holt die Daten von der Diskette in das Feld BA(I), und anschließend wird die Datei wieder ordnungsgemäß geschlossen. Die Zeile 170 erscheint überflüssig, weil die folgende Zeile sowieso die 500 ist. Wenn Sie aber später Zeilen ab 180 einfügen würden, beispielsweise ein Unterprogramm, so vergißt man schnell, den richtigen Sprung zu Zeile 500 (Daten ändern) einzutragen, und dadurch können Fehler im Programm entstehen.

Ab Zeile 500 werden dann die Nummern der Balken und die zugehörigen Daten auf dem Bildschirm ausgegeben. In Zeile 555 kann dann die Nummer des Balkens eingegeben werden, der geändert werden soll. Da es ja keinen "nullten" Balken gibt, kann das Ändern durch Eingabe einer 0 beendet werden. Trotzdem muß in Zeile 565 noch geprüft werden, ob wirklich eine Zahl im erlaubten Bereich eingegeben wurde, denn einen 13. Balken gibt es nicht, und der Versuch, seine Daten in Zeile 570 anzuzeigen, würde zu einer Fehlermeldung führen. Ist N aber richtig eingegeben, dann wird der alte Wert angezeigt, und der neue kann in Zeile 580 eingegeben werden. Anschließend beginnt das Ändern erneut durch Sprung zu Zeile 500.

Dies geht solange, bis eine 0 eingegeben wird. Dann macht das Programm ab Zeile 800 weiter und fragt noch einmal sicherheitshalber nach, ob diese Daten wirklich gespeichert werden sollen. Haben Sie also kein Interesse mehr an den Daten, so genügt eine Eingabe außer J, um das Programm zu beenden. Die eingegebenen Daten sind dann allerdings verloren. Soll aber gespeichert werden, dann wird ein Dateiname erfragt, und in Zeile 830 muß eine eventuell schon vorhandene Datei auf Diskette gelöscht werden. Anschließend kann die Datei dann zum Schreiben geöffnet werden, und die 12 Daten werden in Ihnen nun sicherlich schon bekannter Weise in die Datei geschrieben.

Nach dem Schließen der Datei in Zeile 880 wird noch eine entsprechende Erfolgsmeldung auf dem Bildschirm angezeigt, und das Programm ist zu Ende. Sie sehen, die Datenspeicherung in sequentiellen Dateien ist gar nicht so schwer. Noch einmal zur Erinnerung: Diese Dateien heißen "sequentiell", weil die Daten hintereinander gelesen und geschrieben werden. Sie können also nicht den Wert BA(5) von Diskette holen, bevor Sie nicht BA(4) gelesen haben. Bei der anderen Form der Datenspeicherung als "relative Dateien" ist das dagegen möglich, allerdings kann das nicht Thema eines Einsteigerbuches sein.

7.8 Das Programm BALKENGRAFIK mit Diskettenspeicherung

Jetzt brauchen wir in unser Programm BALKENGRAFIK nur noch ein Unterprogramm zum Einlesen der Daten einzufügen, und schon können wir beliebige "Monatsdateien" von der Diskette einlesen und grafisch anzeigen. Dazu fügen wir eine Zeile 8 ein, löschen die Zeilen 220, 800 und 810 und fügen das Unterprogramm ab Zeile 1000 ein. Das fertige Programm sieht dann folgendermaßen aus:

```
5 DIM MO(12)
8 GOSUB 1000: REM DATEN HOLEN
10 PRINT CHR$(147)
15 PRINT"              MONATSSTATISTIK"
20 FOR I = 1 TO 21
30 PRINT"  ";CHR$(180)
40 NEXT I
50 PRINT"  ";CHR$(108);
60 FOR I = 3 TO 39
70 PRINT CHR$(175);
80 NEXT I
90 PRINT"   JA FE MA AP MA JU JU AU SE OK NO DE";
200 X = 3: X$ = "*"
210 FOR B = 1 TO 12
230 BA% = MO(B) / 100
240 GOSUB 500
250 X = X+3
260 NEXT B
340 GET A$: IF A$ = "" THEN 340
350 PRINT CHR$(147)
```

```
360 END
500 BA% = 22 - BA%
510 FOR I = 22 TO BA% STEP -1
520 Y = I:
530 GOSUB 10000
540 NEXT I
550 RETURN
1000 REM DATEN VON DISKETTE HOLEN
1005 PRINT CHR$(147)
1010 INPUT "BITTE DATEINAMEN EINGEBEN";N$
1020 OPEN5,8,5,N$+",S,R"
1030 FOR I = 1 TO 12
1040 INPUT#5,MO(I)
1050 PRINT MO(I)
1060 NEXT I
1070 CLOSE 5
1080 RETURN
10000 PRINT CHR$(19);
10005 IF Y = 0 THEN 10040
10010 FOR J = 1 TO Y
10020 PRINT CHR$(17);
10030 NEXT J
10040 IF X = 0 THEN 10070
10045 FOR J = 1 TO X
10050 PRINT CHR$(29)
10060 NEXT J
10070 PRINT X$;
10080 RETURN
```

7.9 Zusammenfassung

1. Um konkrete Daten in Programmen verarbeiten zu lassen, hatten wird bisher die folgenden Möglichkeiten:

   Die Eingabe über die Tastatur nach INPUT.

   Die Einzelfestlegung in Programmzeilen.

   Die Sammlung in DATA-Zeilen.

   Bei sequentiellen Dateien werden die Daten separat auf Diskette gespeichert und vom Programm eingelesen. Der Begriff bedeutet, daß die Daten in einer bestimmten Reihenfolge (sequentiell = aufeinander folgend) abgespeichert sind, die zu beachten ist.

2. Zwei zentrale Vorgänge sind bei der Arbeit mit sequentiellen Dateien zu unterscheiden: das Schreiben von Daten auf Diskette und das Einlesen. Für beide Zwecke muß die Datei auf eine entsprechende Weise "geöffnet" werden. Dazu wird der von verschiedenen Vorgängen her bereits bekannte OPEN-Befehl verwendet, wobei hier zwei zusätzliche Hinweise kommen.

   Um eine Datei mit dem Namen MEINEDATEN anzulegen, kann man die folgende Programmzeile verwenden:

```
10 OPEN 5,8,5,"MEINEDATEN,S,W"
```

   Wichtig ist die Zahlengruppe nach dem OPEN, die wir in dieser Form immer verwenden, und außerdem die durch Komma abgetrennten Buchstaben S und W. Das S steht dabei für "sequentiell", und das W steht für WRITE (= Schreiben). Damit weiß der C64, daß er Daten in Richtung Floppy schicken soll.

   Der konkrete Schreibbefehl lautet dann z.B.:

```
50 PRINT#5,X
```

   Dabei ist es wichtig, daß das Leiterzeichen direkt hinter dem PRINT steht und die vom CLOSE-Befehl her bekannte 5 wieder verwendet wird. Das X steht hier für eine numerische Variable. Genauso könnte mit X$ eine Stringvariable auf die Diskette geschrieben werden.

   Nach dem Schreiben der Daten muß der Kanal wieder mit z.B.

```
70 CLOSE 5
```

   geschlossen werden. Wichtig beim "Öffnen" einer Datei zum Schreiben (strenggenommen handelt es sich dabei immer um das "Anlegen" einer Datei) ist, daß nicht schon eine entsprechende Datei auf der Diskette existiert. Sicherheitshalber kann man einen entsprechenden Löschbefehl vorschalten:

```
10 OPEN 1,8,15,"S:MEINEDATEN":CLOSE 1
```

3. Der umgekehrte Vorgang, das Einlesen von Daten von Diskette, funktioniert im Prinzip ähnlich, nur mit entsprechend abgeänderten Befehlen. Auch hier wird zunächst wieder ein Kanal zur Floppy geöffnet, diesmal mit dem Hinweis R für READ (= Lesen).

   ```
   100 OPEN 5,8,5,"S:MEINEDATEN,S,R"
   ```

   Anschließend werden die Daten geholt, z.B.:

   ```
   120 INPUT#5,X
   ```

   Hier sind dieselben Punkte zu beachten wie beim Schreiben der Daten (s.o.).

   Ganz am Schluß muß der Kanal wieder wie oben geschlossen werden, etwa mit:

   ```
   150 CLOSE 5
   ```

4. Wichtig ist, daß man nie versucht, mehr Daten von Diskette zu lesen, als man dort abgelegt hat. Zu diesem Zweck empfiehlt es sich, als erste Zahl immer die Anzahl der Daten abzuspeichern und diese Zahl entsprechend im Programm zu berücksichtigen. Wir haben von dieser Möglichkeit im Programm BALKENGRAFIK nur deshalb keinen Gebrauch gemacht, weil mit den Monaten automatisch die Zahl 12 vorgegeben ist.

8. Die hochauflösende Grafik

Nachdem wir uns schon in Kapitel 6 mit dem Thema Grafik beschäftigt haben, wollen wir uns in diesem Kapitel eine wesentlich flexiblere Welt der Grafik erschließen: die hochauflösende Grafik oder englisch HIRES-Grafik (von High Resolution = hohe Auflösung). Was sind die Besonderheiten dieser Grafik?

Normalerweise arbeitet der C64 im sogenannten Textmodus. In diesem Modus kann er 40 Zeichen pro Zeile und 25 Zeilen darstellen. Dabei ist es ihm völlig gleichgültig, ob ein Zeichen ein "A" oder nur ein "." ist. Wenn Sie sich die Zeichen auf dem Bildschirm einmal genauer anschauen, werden Sie feststellen, daß sie aus einzelnen Punkten zusammengesetzt sind. Dabei stehen für jedes Zeichen 8 Punkte in der Breite und 8 Punkte in der Höhe zur Verfügung. Eigentlich besteht unser 40*25 Zeichen zeigender Bildschirm also aus 40*8 = 320 Punkten waagerecht und 25*8 = 200 Punkten senkrecht. Im hochauflösenden Grafik-Modus können Sie nun jeden dieser 64000 Punkte einzeln setzen oder löschen - eine Arbeit, die wir natürlich am liebsten wieder den C64 machen lassen.

8.1 Die Besonderheiten der hochauflösenden Grafik

Wir möchten Ihnen zu Beginn ein wenig die grundsätzliche Funktion der Grafik auf dem C64 erklären, damit Sie die folgenden Vorgänge besser verstehen können.

Modus I: Der Textmodus

Der C64 enthält ein Bauteil, den sogenannten Video-Controller (von uns im folgenden VIC genannt), der sich komplett um die Grafik kümmert. Normalerweise arbeitet er im Textmodus. Dabei können auf dem Bildschirm entsprechend den 40 Spalten und 25 Zeilen 1000 Zeichen dargestellt werden, wie Sie leicht mit dem Cursor er"fahren" können. Die 1000 Zeichen liegen als Nummern in einem ebenso großen Speicherbereich des C64, dem Bildschirmspeicher. Das heißt, das "A", das Sie auf dem Bild-

schirm sehen, ist im Bildschirmspeicher des C64 etwa als "1" abgelegt. Wenn Sie also einfach Ihren Bildschirm ausschalten, verschwindet zwar das "A", aber die zugehörige "1" bleibt erhalten. Wenn Sie jetzt Ihren Bildschirm wieder einschalten, kommt der VIC ins Spiel. Er schaut in allen tausend Speicherstellen nach, und unter anderem wird dabei aus der "1" an der richtigen Stelle wieder das "A".

Dabei holt er sich das Aussehen für den Bildschirm (also die 8*8-Punkte-Information) aus einem anderen Speicherbereich, der fest vorgegeben ist. Diese 64 Punkte zeigt er auf dem Fernseher oder Monitor in der richtigen Form an.

Modus II: Die hochauflösende Grafik

Wollen wir nun die hochauflösende Grafik verwenden, so müssen wir dies dem VIC mitteilen. Von dem Augenblick an holt er nicht mehr die fertigen Zeichen für die 1000 Speicherplätze, sondern schaut in einem 64000 Punkte großen Bereich nach, welche Punkte gesetzt und welche gelöscht sind, und stellt diese Punkte auf dem Bildschirm dar. Dabei werden die bisher auf dem Bildschirm vorhandenen Zeichen natürlich nicht mehr dargestellt, denn beides gleichzeitig kann der VIC nicht. Dadurch verlassen wir also den sicheren Boden unserer bisherigen Bildschirmdarstellung. Wir können kein BASIC-Programm mehr auflisten, Änderungen vornehmen, und bei der Betätigung von Tasten sehen wir nichts mehr. Wir sollten also am Ende eines Programms unbedingt wieder in den Text-Modus zurückschalten.

Ergänzung: Die Farben

Auf eine weitere Besonderheit müssen wir noch kurz zu sprechen kommen. Sicherlich wissen Sie schon, daß Ihr C64 16 Farben gleichzeitig darstellen kann. Allerdings ist die Farbe für jeweils ein 8*8-Punkte-Quadrat gleich, somit können Sie also jeden Buchstaben auf dem Bildschirm in einer anderen Farbe darstellen. Die Information für die Farbe eines solchen Quadrats muß sich Ihr Computer natürlich auch merken - er tut das in einem ebenfalls 1000 Speicherplätze großen Bereich, so daß für

jedes Zeichen ein Speicherplatz bereitgehalten wird und für die Farbe ebenfalls. Und jetzt wird es wirklich verwirrend: Wenn wir den VIC auf die hochauflösende Grafik umschalten, holt er sich die Farbe für die Quadrate nicht mehr aus dem bisherigen Farbbereich, sondern aus dem Bereich, in dem vorher die 1000 Zeichen lagen.

Nach soviel Theorie wollen wir aber nun endlich zu unserer ersten Grafik kommen. Trotzdem waren diese Vorbemerkungen unserer Meinung nach notwendig, damit Sie nicht beispielsweise während der Darstellung einer hochauflösende Grafik die RUN/STOP-Taste betätigen und dadurch nicht mehr in den Text-Modus zurückkommen, denn der rettende Programmteil, in dem das gemacht wird, wird ja im Programm wegen des Abbruchs nicht mehr erreicht.

8.2 Ein- und Ausschalten der hochauflösenden Grafik

Wie kommen wir denn nun in diese hochauflösende Grafik hinein? Am besten geben Sie erst einmal das folgende kleine Programm ein und schauen sich das Ergebnis an:

```
10 HI = 53265
20 AD = 53272
30 POKE HI,187
40 POKE AD,29
50 FOR I = 1 TO 1000: NEXT I
80 POKE HI,27
90 POKE AD,21
100 END
```

Wenn Sie dieses Programm starten, so werden Sie wahrscheinlich eine bange Sekunde lang glauben, wir hätten Ihren C64 kaputt gemacht. Plötzlich haben Sie ein eigenartiges Punktemuster auf dem Bildschirm und - wenn Sie einen Farbfernseher oder Farbmonitor benutzen - auch noch farbige Kästchen, dann taucht der gewohnte Bildschirm wieder auf. Wir hoffen, daß Sie keinen Fehler beim Abtippen gemacht haben, denn wenn Ihr C64 dieses Programm mit einem Syntax-Error abbricht, bleibt die hochauflösende Grafik eingeschaltet. Zurück in den gewohnten Text-

Modus kommen Sie, indem Sie gleichzeitig die RUN/STOP-Taste und die RESTORE-Taste drücken.

Das Punktemuster, das Sie gesehen haben, rührt daher, daß in dem 64000 Punkte großen Bereich irgendwelche Punkte gesetzt sind und diese jetzt auf dem Bildschirm dargestellt werden. Es kann sich nach dem Einschalten des C64 um zufällige Punkte handeln, oder es sind die Reste früherer Arbeiten im Grafikmodus. Später werden wir diesen Bereich natürlich vor dem Umschalten "saubermachen". Aber woher kommen die farbigen Kästchen beziehungsweise bei Schwarz-Weiß-Bildschirmen unterschiedliche Graustufen? Das können wir ganz einfach herausfinden. Löschen Sie den Bildschirm, und lassen Sie sich mit LIST das Programm auflisten. Nun merken Sie sich bitte die Lage der Zeichen auf dem Bildschirm und starten das Programm.

Ist Ihnen etwas aufgefallen? Dort, wo Zeichen auf dem Bildschirm waren, sind nach dem Umschalten farbige Kästchen und umgekehrt. Wir hatten ja schon bei den Besonderheiten der hochauflösenden Grafik darauf hingewiesen, daß der VIC in diesem Modus den Bereich, in dem sonst die Zeichen abgelegt sind, für die Farbbestimmung nutzt und so versucht, aus jedem "Zeichen" - diese stehen dort weiterhin - eine Farbe zu machen. Der einfachste Weg zu einem einfarbigen Bildschirm stellt das Löschen des Bildschirms dar. Ändern Sie deshalb bitte das Programm in der folgenden Weise ab (dabei entspricht in der Zeile 60 das PRINT CHR$(147) dem früher verwendeten PRINT "CLR", wobei CLR für die Clr/Home-Taste, hier verbunden mit Shift, stand):

```
10 HI = 53265
20 AD = 53272
30 POKE HI,187
40 POKE AD,29
50 FOR I = 1 TO 1000: NEXT I
60 PRINT CHR$(147)
70 FOR I = 1 TO 1000: NEXT I
80 POKE HI,27
90 POKE AD,21
100 END
```

Nun wird in Zeile 60 der Bildschirm gelöscht, und anschließend haben Sie durch die Warteschleife in Zeile 70 noch einmal eine Sekunde Zeit, den einfarbigen Bildschirm zu betrachten. Wir möchten nicht im einzelnen erklären, wie das Umschalten zwischen den Grafikmodi funktioniert. Zum Schluß dieses Kapitels stellen wir Ihnen lieber ganz kurz ein Buch vor, das auf diese und viele andere Fragen ausführlich Antwort gibt. Wichtig ist jedoch, daß Sie mit den Zeilen 10 - 40 die hochauflösende Grafik einschalten und mit den Zeilen 80 - 90 wieder ausschalten können.

8.3 Sauberer Bildschirm bei hochauflösender Grafik

Nun haben wir zwar einen einfarbigen Bildschirm, aber das Punktemuster ist ja immer noch vorhanden. Um einen wirklich "sauberen" Bildschirm zu erhalten, müssen wir 64000 Punkte löschen. Natürlich können wir dazu wieder gut den C64 einsetzen, aber erst einmal müssen wir wissen, wo diese Punkte sind und wie sie gelöscht werden.

Diese Punkte beginnen im Speicher des C64 ab der Speicherstelle 8192. Wenn Sie sich jetzt wundern, woher wir das so genau wissen, nur ein kurzer Hinweis. Die ca. 64000 Speicherstellen des C64 (daher C64) haben eine forlaufende Nummer und sind zweitens in große Bereiche eingeteilt, die für verschiedene Aufgaben zuständig sind. Der Bereich für die hochauflösende Grafik beginnt eben bei 8192 und geht bis 16192. Dabei liegen jeweils 8 Punkte in einer Speicherstelle. Wenn wir eine solche Speicherstelle auf 0 setzen, werden also die 8 Punkte gelöscht. Um einen sauberen Bildschirm zu bekommen, müssen wir also ab 8192 insgesamt 8000 Speicherstellen auf 0 setzen. Das folgende Programm erledigt diese Aufgabe für Sie, und Sie werden nach dem Starten feststellen, daß Ihr C64 da schon ganz schön was zu tun bekommt:

```
10 HI = 53265
20 AD = 53272
30 POKE HI,187
40 POKE AD,29
50 PRINT CHR$(147)
```

```
60 FOR I = 1 TO 8192 TO 16192
70 POKE I,0
80 NEXT I
90 GET A$: IF A$ = "" THEN 90
100 POKE HI,27
110 POKE AD,21
120 END
```

Es sieht so aus, als würde unser Programm den Bildschirm "sauberfegen". Dazu benötigt der C64 immerhin 45 Sekunden. Sollte bei Ihnen auf dem Bildschirm nichts passieren, so war ihr Bildschirm eventuell schon gelöscht und Sie sehen vom Löschvorgang nichts mehr. Dies ist zum Beispiel beim mehrmaligen Starten des Programms der Fall.

Das Programm endet erst, wenn Sie eine Taste drücken. Das liegt an der eigenartigen Programmzeile 90. Bei ihr handelt es sind um eine sogenannte Endlosschleife. Solange eine bestimmte Bedingung (A$ = "") erfüllt ist, bleibt der Rechner in derselben Programmzeile stecken (THEN 90).

Woran merkt der C64, daß Sie eine Taste gedrückt haben? Dafür ist der GET-Befehl zuständig. Er schaut nach, ob inzwischen eine Taste gedrückt wurde. In so einem Fall bringt er das Zeichen mit und legt es in der Variablen A$ ab. Andernfalls bringt er nichts zurück. Nichts ist bei einer Stringvariablen eben "" und so muß er (THEN 90) noch mal los (GET A$), und daraus ergibt sich die Endlosschleife, die Sie nur durch ein "anständiges" A$ (einen Tastendruck) beenden können. Nun ändern Sie doch einmal die Zeile 70 in

```
70 POKE I,15
```

um und starten das Programm erneut. Sie sehen, daß durch diese kleine Änderung etwas ganz anderes auf dem Bildschirm herauskommt. Durch eine 0 löschen wir alle 8 zusammengehörenden Punkte, und durch eine 15 setzen wir einige davon. Allerdings wollen wir ja noch einiges mit der hochauflösenden Grafik machen, und dazu müssen wir wieder die alte Zeile

```
70 POKE I,0
```

herstellen und noch eine kleine Änderung an unserem Programm vornehmen. Wenn wir mit Zeile 50 den Bildschirm löschen, so haben wir eigentlich nicht den richtigen Farbwert für unsere Grafik. Deshalb ändern wir diese Zeile folgendermaßen ab:

```
50 FOR I = 1024 TO 2023: POKE I,1 : NEXT I
```

Wenn Sie das so geänderte Programm starten, sehen Sie, daß der C64 erst sehr schnell die Farben auf den richtigen Wert setzt und dann etwas langsamer die 64000 Punkte. Speichern Sie das Programm bitte in dieser Form unter dem Namen "HiresInit" auf der Diskette ab. Wir werden es nämlich in diesem Kapitel noch mehrmals als Basis für weitere Programme verwenden. Übrigens ist am Ende des Programms der Bildschirm voller "M", die Sie leicht durch PRINT "CLR" wegbekommen.

8.4 Punkte setzen

Nachdem wir nun den Bildschirm problemlos umgeschaltet und leergeräumt haben, müssen wir jetzt sehen, wie wir die einzelnen kleinen Punkte auf den Bildschirm bekommen, aus denen Grafik nun einmal besteht. An dieser Stelle stecken wir in einem echten Dilemma. Einerseits möchten wir Ihnen gerne erklären, wie die 64000 Punkte im Speicher liegen und wie die 8 Punkte einer Speicherzelle beschaffen sind, damit Sie überhaupt mit der Grafik arbeiten können, andererseits würde das viele Seiten dieses Buches füllen, und es gehört auch nicht in ein Einsteigerbuch. Deshalb haben wir uns folgende Lösung überlegt. Wir drucken einfach ein Unterprogramm ohne weitere Erklärungen ab, das die gewünschte Arbeit leistet. Wenn Sie dann weitere Informationen wünschen, können Sie sich entsprechende Literatur besorgen, oder Sie können einfach dieses Unterprogramm in eigenen Programmen einsetzen.

Dieses Unterprogramm haben wir ab Zeile 50000 abgelegt, so daß es immer am Ende unserer Programme liegt und nicht stört oder versehentlich abgearbeitet wird. Dieses Unterprogramm benötigt von Ihnen natürlich Angaben darüber, wo dieser Punkt genau liegen soll. Zu diesem Zweck haben wir einfach die Punkte von links nach rechts (X-Koordinate) und von oben nach

unten (Y-Koordinate) durchnumeriert. Diesem Unterprogramm übergeben wir später in XX die gewünschte X-Koordinate zwischen 0 und 319 und in YY die gewünschte Y-Koordinate zwischen 0 und 199. Die Zahlen ergeben sich aus folgender Überlegung: Da in eine Zeile im Textmodus 40 Zeichen gehen und jedes Zeichen 8 Punkte breit ist, passen 320 Punkte (0 - 319) in eine Zeile. Die 25 Bildschirmzeilen mit je 8 Punkten Höhe ergeben 200 Punkte in der vertikalen Richtung (0 - 199).

Das Unterprogramm setzt einen durch die beiden Koordinaten bestimmten Punkt an die richtige Stelle. Dabei benutzt es selbst nur Variablennamen, die mit Z anfangen. Achten Sie also bitte darauf, diese Variablennamen in eigenen Programmen nicht zu benutzen, da sie sonst unkontrolliert geändert werden. Am besten laden Sie noch einmal unser Programm "HiresInit" und fügen dann die folgenden Zeilen an. Denn meist werden Sie ja das Einschalten und Löschen der Grafik (HiresInit) und das Unterprogramm zum Punktesetzen gleichzeitig verwenden wollen.

Das notwendige Unterprogramm sieht folgendermaßen aus:

```
50000 REM PUNKT SETZEN
50010 IF XX < 0 OR XX > 319 THEN RETURN
50020 IF YY < 0 OR YY > 199 THEN RETURN
50030 Z1% = YY/8
50040 Z2% = YY-Z1%*8
50050 Z = 8192 + 320 * Z1% +Z2%
50060 Z1% = XX/8
50070 Z2% = XX - Z1%*8
50080 Z = Z + Z1% * 8
50090 POKE Z,PEEK(Z) OR 2 ^ (7-Z2%)
50100 RETURN
```

In Zeile 50090 ist hinter "OR 2" der Hochpfeil über der Return-Taste gemeint.

Beachten Sie bitte, daß der Nullpunkt beim Gebrauch unseres Unterprogramms immer oben links liegt, dort liegt also der Punkt mit den Koordinaten XX=0 und YY=0. Nach rechts wird der XX-Wert größer und nach unten der YY-Wert. Speichern Sie dieses neue Gesamtprogramm nun bitte unter dem Namen "HiresPunkt" auf der Diskette ab. Nun wollen wir die Routine

einmal dazu einsetzen, einen waagerechten Strich in der Bildschirmmitte zu ziehen. Wir brauchen also nur YY auf 100 (Bildschirmmitte) zu setzen und XX von 0 (linker Bildschirmrand) bis 319 zu verändern. Das dazu notwendige Gesamtprogramm entsteht einfach durch Zufügen der Zeile 85 und der Zeilen 200-260 und sieht folgendermaßen aus:

```
10 HI = 53265
20 AD = 53272
30 POKE HI,187
40 POKE AD,29
50 FOR I = 1024 TO 2023: POKE I,1 : NEXT I
60 FOR I = 8192 TO 16192
70 POKE I,0
80 NEXT I
85 GOSUB 200: REM HAUPTPROGRAMM
90 GET A$: IF A$ = "" THEN 90
100 POKE HI,27
110 POKE AD,21
120 END
200 REM HAUPTPROGRAMM
210 GET A$: IF A$ = "" THEN 210
220 YY = 100
230 FOR XX = 0 TO 319
240 GOSUB 50000: REM PUNKT SETZEN
250 NEXT XX
260 RETURN
50000 REM PUNKT SETZEN
50010 IF XX < 0 OR XX > 319 THEN RETURN
50020 IF YY < 0 OR YY > 199 THEN RETURN
50030 Z1% = YY/8
50040 Z2% = YY-Z1%*8
50050 Z = 8192 + 320 * Z1% +Z2%
50060 Z1% = XX/8
50070 Z2% = XX - Z1%*8
50080 Z = Z + Z1% * 8
50090 POKE Z,PEEK(Z) OR 2 ^ (7-Z2%)
50100 RETURN
```

Es dauert natürlich wieder 45 Sekunden, bis der Bildschirm gelöscht ist. Da er wahrscheinlich vom letzten Programm noch leer ist, sehen Sie von der anstrengenden Arbeit des C64 herzlich wenig. Sie können ja in der Zeit schon mal überlegen, welche Taste Sie nehmen wollen, um das anschließende Ziehen des

Strichs auszulösen. Am Schluß wird das Programm durch das Drücken einer Taste beendet, und der Bildschirm ist voller "M".

8.5 Ein Mini-Malprogramm

Mit unseren bisherigen Kenntnissen und vorliegenden Programmteilen können wir nun schon problemlos ein kleines Malprogramm erstellen.

Wir realisieren dazu die Funktion "Freihandzeichnen", die in den meisten fertigen Programmen vorhanden ist. Dabei kann man mit den Cursor-Tasten einen kleinen Punkt auf dem Bildschirm bewegen und damit gleichzeitig eine Linie ziehen. Denken Sie bitte daran, daß es auch hier wieder 45 Sekunden dauert, bis Sie beginnen können.

```
10 HI = 53265
20 AD = 53272
30 POKE HI,187
40 POKE AD,29
50 FOR I = 1024 TO 2023: POKE I,1 : NEXT I
60 FOR I = 8192 TO 16192
70 POKE I,0
80 NEXT I
85 GOSUB 200: REM HAUPTPROGRAMM
90 GET A$: IF A$ = "" THEN 90
100 POKE HI,27
110 POKE AD,21
120 END
200 REM HAUPTPROGRAMM
210 YY = 100: XX = 180
220 GOSUB 50000: REM ERSTEN PUNKT SETZEN
230 GET A$: IF A$ = "" THEN 230
240 IF A$ = CHR$(17)  THEN YY = YY + 1: REM UNTEN
250 IF A$ = CHR$(145) THEN YY = YY - 1: REM OBEN
260 IF A$ = CHR$(29)  THEN XX = XX + 1: REM RECHTS
270 IF A$ = CHR$(157) THEN XX = XX - 1: REM LINKS
280 IF A$ = CHR$(81)  THEN RETURN: REM Q = ENDE
290 GOSUB 50000
300 GOTO 230
50000 REM PUNKT SETZEN
50010 IF XX < 0 OR XX > 319 THEN RETURN
50020 IF YY < 0 OR YY > 199 THEN RETURN
```

```
50030 Z1% = YY/8
50040 Z2% = YY-Z1%*8
50050 Z = 8192 + 320 * Z1% +Z2%
50060 Z1% = XX/8
50070 Z2% = XX - Z1%*8
50080 Z = Z + Z1% * 8
50090 POKE Z,PEEK(Z) OR 2 ^ (7-Z2%)
50100 RETURN
```

Kleiner Trick - große Hilfe: Programmpuzzle

Dieses Programm ist einfach durch Änderung des Hauptprogramms entstanden. Bevor wir auf die neuen Zeilen eingehen, wollen wir Sie noch kurz darauf hinweisen, daß man sich auf diese Weise viel Arbeit sparen kann. Man schreibt einfach Teile des Programms so, daß man sie in mehreren anderen Programmen weiterverwenden kann. In unserem Fall sind das die Teile zum Ein- und Ausschalten der hochauflösenden Grafik und zum Setzen eines Punktes. Nun aber zu den neu hinzugekommenen Zeilen:

Erklärung des Programms

In Zeile 210 werden die Anfangskoordinaten für unseren Punkt auf die Bildschirmmitte gesetzt. Dann wird in Zeile 220 an dieser Stelle der Punkt auf den Bildschirm geschrieben. In Zeile 230 wird in bekannter Weise auf das Drücken einer Taste gewartet, das entsprechende Zeichen befindet sich dann in der Variablen A$. Nun wird in Zeile 240 geprüft, ob dieses Zeichen der Cursor-Down-Taste entsprach. In diesem Fall wird einfach zur Y-Position eins dazugezählt. Sie erinnern sich doch noch daran, daß CHR$ das Zeichen zu einer Zahl liefert? Entsprechend werden in den Zeilen 250 bis 270 für die drei anderen Cursor-Tasten die Koordinaten geändert. Damit Sie das "Freihandzeichnen" auch beenden können, wird in Zeile 280 geprüft, ob die gedrückte Taste ein Q für QUIT war. In diesem Fall wird das Hauptprogramm mit RETURN beendet und dadurch die hochauflösende Grafik ausgeschaltet. Ansonsten wird in Zeile 290 der Punkt an der neuen Stelle gesetzt, und das Ganze beginnt erneut durch den Sprung zu Zeile 230.

Vielleicht fragen Sie sich, woher wir die Zahlen für die Cursor-Tasten haben. Nun, zum einen gibt es im Handbuch eine sogenannte ASCII-Tabelle, in der man für die Tasten diese Werte nachschlagen kann. Einfacher geht es aber mit folgendem zweizeiligen BASIC-Programm:

```
10 GET A$: IF A$ = "" THEN 10
20 PRINT ASC(A$): GOTO 10
```

Starten Sie einfach dieses Programm, und drücken Sie die gesuchte Taste, sofort wird deren zugehörige Zahl auf dem Bildschirm angezeigt. Wenn Sie alle Tasten einmal durchprobiert haben, können Sie das Programm durch Drücken der RUN/STOP-Taste beenden.

Übrigens paßt unsere Unterroutine zum Zeichnen eines Punktes darauf auf, daß der Punkt innerhalb des erlaubten Bereiches bleibt. Liegt er außerhalb, so wird einfach kein Punkt gesetzt. Das ist sehr wichtig, weil es uns der C64 etwas übel nimmt, wenn wir einfach "irgendwo" Punkte setzen.

8.6 Mathematik in höchster Auflösung

Haben Sie auch manchmal Schwierigkeiten, sich mathematische Funktionen und Formeln vorzustellen? Dann ist dieser Abschnitt genau richtig für Sie, denn die hochauflösende Grafik ist ideal zur Darstellung von Funktionen geeignet. Wir wollen Ihnen das einmal am Beispiel einer Sinus-Kurve zeigen.

Dazu wollen wir ein paar Vorüberlegungen anstellen. Der Sinus wiederholt sich jeweils nach 2 PI, also nach 6,28. Wenn wir also den Sinus einfach entlang unserer X-Achse auftragen würden, hätten wir also ungefähr 51mal den SINUS und zwar ganz "schmal gezeichnet". Wir wollen aber lieber nur einmal den kompletten Sinus auf dem Bildschirm darstellen, weil er dann viel genauer aufgelöst ist. Also benutzen wir als Formel nicht SIN(XX), sondern SIN(XX/50).

Auch für die Y-Werte müssen wir uns erst eine geeignete Umrechnung überlegen. Würden wir einfach den Wert der SINUS-

Funktion nehmen (YY = SIN(XX/50), so kämen nur Werte zwischen -1 und +1 heraus. Sie können das leicht nachprüfen, indem Sie im Direktmodus schreiben:

```
FOR XX = 0 TO 6.3 STEP 0.3: PRINT SIN(XX): NEXT XX
```

Außerdem würden die negativen YY-Werte nicht angezeigt, weil sie außerhalb des erlaubten Bildschirms liegen. Also addieren wir in jedem Fall schon einmal 100 zu YY hinzu, damit das Zeichnen in der Mitte des Bildschirms beginnt. Zusätzlich multiplizieren wir den SINUS mit 99, so daß als kleinster Wert 100-1*99 = 1 und als größter Wert 100+1*99 = 199 entsteht. Dadurch bleiben die YY-Werte in jedem Fall im erlaubten Bereich. Mit diesen Vorüberlegungen ist es nun ganz einfach, die nötigen BASIC-Zeilen hinzuschreiben. Bitte benutzen Sie zur Arbeitserleichterung wieder das Programm "HiresPunkt", und fügen Sie nur die notwendigen Zeilen hinzu:

```
10 HI = 53265
20 AD = 53272
30 POKE HI,187
40 POKE AD,29
50 FOR I = 1024 TO 2023: POKE I,1 : NEXT I
60 FOR I = 8192 TO 16192
70 POKE I,0
80 NEXT I
85 GOSUB 200: REM HAUPTPROGRAMM
90 GET A$: IF A$ = "" THEN 90
100 POKE HI,27
110 POKE AD,21
120 END
200 REM HAUPTPROGRAMM
210 FOR XX = 0 TO 319
220 Y = SIN(XX/50)
230 YY = 100-Y*99
240 GOSUB 50000
250 NEXT XX
260 RETURN
50000 REM PUNKT SETZEN
50010 IF XX < 0 OR XX > 319 THEN RETURN
50020 IF YY < 0 OR YY > 199 THEN RETURN
50030 Z1% = YY/8
50040 Z2% = YY-Z1%*8
50050 Z = 8192 + 320 * Z1% +Z2%
```

```
50060 Z1% = XX/8
50070 Z2% = XX - Z1%*8
50080 Z = Z + Z1% * 8
50090 POKE Z,PEEK(Z) OR 2 ^ (7-Z2%)
50100 RETURN
```

Vielleicht wundert Sie noch die Zeile 230, weil wir Y*99 von 100 abziehen. Sie können ja einfach mal ein +Y*99 daraus machen, dann sehen Sie sofort den Unterschied: In einem üblichen Koordinatensystem liegen Punkte mit größerem Y-Wert nämlich höher, in unserem System aber niedriger. Deshalb setzen wir ein Minus-Zeichen davor und drehen damit unser Koordinatensystem "auf den Kopf".

Wie man andere Funktionen auf den Bildschirm bekommt

Durch Ändern der Zeilen 220 und 230 können Sie nun auch recht einfach andere Funktionen darstellen. Am einfachsten ist noch der Cosinus, weil Sie denselben Bereich der Zahlen verwenden können. Die geänderten Zeilen lauten damit einfach:

```
220 Y = COS(XX/50)
230 YY = 100-Y*99
```

Wenn Sie andere mathematische Funktionen ausprobieren wollen, so sollten Sie sich überlegen, welche Werte dabei von 0 bis 319 entstehen und entsprechend geeignete Faktoren verwenden. Nehmen wir als Beispiel die Quadrat-Funktion Y = X*X. Würden wir diese Funktion so benutzen, so käme für X = 319 ein Y-Wert von 101761 heraus. Sie können ja mal probeweise folgende beiden Zeilen benutzen:

```
220 Y = XX * XX
230 YY = 100-Y/1020
```

Sie sehen, daß das Ergebnis nicht gerade aussagekräftig ist. Übrigens sind wir auf die Formel Y/1020 gekommen, indem wir den größten Y-Wert, der entstehen kann (101761), durch die erlaubten 100 geteilt haben und dabei durch Aufrunden auf 1020 kamen.

Probieren Sie doch einmal die folgenden drei Zeilen aus, die die Funktion schon erheblich aussagekräftiger auf den Bildschirm bringen:

```
220 X = (XX - 160)/5
225 Y = X * X
230 YY = 100 - Y/11
```

8.7 Zusammenfassung

1. Zu unterscheiden sind beim C64 der Textmodus und der Grafikmodus (= hochauflösende oder HIRES-Grafik). Der Textmodus ist nach dem Einschalten des Rechners sofort eingestellt. In diesem Modus kann der C64 40 Zeichen pro Zeile und 25 Zeilen darstellen, insgesamt also 1000 Zeichen. Beim Druck einer Taste oder Tastenkombination (etwa A-Taste plus Shift-Taste) holt sich der dafür zuständige Video-Controller aus einem entsprechenden Speicherbereich die Informationen, die er braucht, um das Zeichen darzustellen. Jedes Zeichen besteht ja in Wirklichkeit aus 8*8 Kleinpunkten. Und in dieser Form ist es im Speicher reserviert.

2. Im Grafikmodus kann man über die insgesamt 64000 Kleinpunkte, aus denen alle Textmodus-Zeichen zusammengesetzt sind, einzeln verfügen. Dafür gibt es einen eigenen Speicherbereich, in dem der Video-Controller jederzeit feststellen kann, welche Punkte gesetzt und welche leer sind.

3. Beide Modi, der Textmodus und der Grafikmodus, sind nicht mischbar. Aus diesem Grunde muß man eindeutig auf hochauflösende Grafik umschalten: Das geschieht mit den folgenden Programmzeilen:

   ```
   10 HI=53265
   20 AD=53272
   30 POKE HI,187
   40 POKE AD,29
   ```

   Zurückgeschaltet in den Textmodus wird z.B. mit:

```
80 POKE HI,27
90 POKE AD,21
```

4. Die Einzelpunkte der hochauflösenden Grafik liegen beim C64 im Speicherbereich zwischen 8192 und 16192. Dabei liegen immer acht zusammen, so daß man insgesamt wieder auf 64000 kommt. Um einen "sauberen" Bildschirm zu bekommen, müssen ab 8192 8000 Speicherstellen auf Null gesetzt werden. Das geschieht z.B. durch den folgenden Programmausschnitt:

```
60 FOR I = 8192 TO 16192
70 POKE I,0
80 NEXT I
```

   Durch eine Änderung der 0 in einen anderen Wert, etwa eine 15, bekommt man ein entsprechendes Punktemuster auf dem Bildschirm.

5. Einen einzelnen Punkt kann man in der hochauflösenden Grafik durch das folgende Unterprogramm bekommen, das Sie (ohne es im einzelnen verstehen zu müssen) in Ihre Programme einbauen können:

```
50000 REM PUNKT SETZEN
50010 IF XX < 0 OR XX > 319 THEN RETURN
50020 IF YY < 0 OR YY > 199 THEN RETURN
50030 Z1% = YY/8
50040 Z2% = YY-Z1%*8
50050 Z = 8192 + 320 * Z1% +Z2%
50060 Z1% = XX/8
50070 Z2% = XX - Z1%*8
50080 Z = Z + Z1% * 8
50090 POKE Z,PEEK(Z) OR 2 ^ (7-Z2%)
50100 RETURN
```

   In Zeile 50090 ist hinter "OR 2" der Hochpfeil über der Return-Taste gemeint.

   Es empfiehlt sich beim BASIC des C64 sowieso, häufig benötigte Programmteile auf ganz bestimmte Zeilennummern zu legen, die man nur dafür benutzt. Dann kann man diese Teilprogramme in andere hineinpuzzeln. Man muß nur darauf achten, daß in dem zugeladenen Teilprogramm

keine Variablen vorkommen, die man zu anderem Zweck im übrigen Programm verwendet. Das würde zu unliebsamen Überraschungen führen.

6. Wenn man einzelne Programmteile langsamer ablaufen lassen will, um sie etwa in Ruhe zu betrachten, kann man sogenannte Verzögerungsschleifen verwenden. Dabei läßt man den C64 einige Zeit lang "unnötige" Arbeit verrichten, damit wir als Benutzer Schritt halten können. Eine solche Warteschleife könnte etwa so aussehen:

```
10 PRINT "ICH FANGE MIT DER WARTESCHLEIFE AN"
20 FOR I = 1 TO 1000
30 NEXT I
40 PRINT "ICH BIN FERTIG"
```

Mit dieser Warteschleife ist der C64 etwa anderthalb Sekunden beschäftigt. Man kann sie für die eigenen Bedürfnisse entsprechend ändern.

7. Daneben gibt es noch die sogenannte Endlosschleife. Sie liegt etwa in der folgenden Zeile vor:

```
90 GET A$: IF A$ = "" THEN 90
```

Hier arbeitet der C64 die Schleife solange ab, d.h. er prüft, ob eine Taste gedrückt wird, bis das endlich erfolgt und er weiterarbeiten kann.

9. Sprites

Nachdem wir nun schon zwei verschiedene Möglichkeiten kennengelernt haben, Grafik auf dem Bildschirm zu erzeugen (ASCII-Zeichen-Grafik und hochauflösende Grafik), wollen wir Ihnen nun noch eine weitere Möglichkeit zeigen, die viele Vorteile hat und den C64, nebenbei bemerkt, weltberühmt gemacht hat: Sprites.

9.1 Was sind Sprites?

Sprites sind kleine Teile einer Grafik, die auf sehr einfache Weise schnell bewegt werden können. Damit eignen sie sich hervorragend für Simulationen und Spiele. Dabei können Sprites sogar in beiden anderen Grafikarten benutzt werden, also in der ASCII-Zeichen-Grafik und der hochauflösenden Grafik. Das Aussehen der Sprites muß natürlich irgendwo festgelegt werden. Dies geschieht nicht in den 1000 Speicherplätzen für die Zeichengrafik, auch nicht in den 8192 Speicherplätzen für die hochauflösende Grafik, sondern in einem speziellen Bereich. Dadurch wird auch schon deutlich, daß Sprites immer zusätzlich zur gerade angezeigten Grafik erscheinen.

Sprites können 24 HiRes-Punkte breit und 21 Punkte hoch sein. Da 8 Punkte in der Breite genau in einen Speicherplatz passen, benötigen wir insgesamt 24/8 = 3 Speicherplätze in der Breite. Die 21 vertikalen Punkte verbrauchen mehr Speicherplätze, nämlich eben 21. Warum das so ist, wollen wir an dieser Stelle nicht näher erklären. 3 * 21 ergibt gerade 63 Speicherplätze. Bevor wir Ihnen nun weitere Details zu Sprites erklären und Ihnen die Programmierung näherbringen, wollen wir doch wenigstens versuchen, eins dieser geheimnisvollen Objekte auf dem Bildschirm zu präsentieren.

9.2 Das erste Sprite

Geben Sie das folgende Programm ein, und speichern Sie es anschließend unter "Erstes Sprite" auf der Diskette ab.

```
10 REM "ERSTES SPRITE-PROGRAMM"
20 PRINT CHR$(147)
30 V = 53248
40 POKE V,50
50 POKE V+1,50
60 POKE V+21,1
70 GET A$: IF A$ = "" THEN 70
80 POKE V+21,0
90 END
```

Obwohl Sie sicherlich darauf brennen, das Programm sofort zu starten, wollen wir es Ihnen erst einmal kurz erklären. In Zeile 20 wird der Bildschirm gelöscht. In Zeile 30 sparen wir uns durch die Variablenzuweisung viel Schreibarbeit. Diese schlecht zu merkende Zahl stellt eine magische Adresse bei der Arbeit mit Sprites dar. Hinter ihr verbirgt sich nämlich der "Briefkasten" des VIC (Video-Controllers), und dort müssen wir die Anweisungen hinterlassen, was er mit den Sprites tun soll.

Die Zeilen 40 - 60 stellen Anweisungen dar, wo das Sprite auf dem Bildschirm positioniert werden soll (in diesem Fall ziemlich oben links) und daß es auch wirklich auf dem Bildschirm erscheinen soll, denn es kann auch unsichtbar an einer Stelle des Bildschirms positioniert sein. Der Befehl POKE ermöglicht es dem Benutzer des C64, in jede der ca. 64000 Speicherstellen einen Wert direkt hineinzuschreiben. Dieser Wert muß im erlaubten Bereich zwischen 0 und 255 liegen. Seien Sie bei eigenen Experimenten äußerst vorsichtig, denn für den geordneten Betrieb des C64 müssen bestimmte Speicherstellen unverändert bleiben, sonst verweigert er Ihnen seine weitere Mitarbeit. Sie erinnern sich ja noch an den folgenschweren POKE-Befehl, mit dem Sie auf die hochauflösende Grafik umgeschaltet haben.

In Zeile 80 wird nach Drücken einer Taste das gezeigte Sprite wieder ausgeschaltet.

Wenn Sie nun das Programm starten, sollten Sie oben links auf dem Bildschirm ein eigenartiges Objekt entdecken, das nach dem Drücken einer Taste wieder verschwindet: unser erstes Sprite. Sollte auf Ihrem Bildschirm gar nichts erscheinen, so beenden Sie bitte das Programm durch Tastendruck und geben ein:

```
POKE 2040,32
```

Wenn Sie jetzt das Programm erneut starten, müßten Sie endlich auch zu Ihrem Sprite kommen.

Nun, zugegeben: Sehr schön sieht das Ganze noch nicht aus, aber wir sind ja auch erst am Anfang und haben das Aussehen des Sprites ja noch gar nicht festgelegt. Es ist genauso zufällig wie die dargestellte hochauflösende Grafik nach dem Einschalten in Kapitel 8. Trotzdem können wir am Beispiel dieses Sprites schon die meisten Besonderheiten zeigen. Ändern Sie deshalb das vorhergehende Programm ab, indem Sie eine Zeile

```
65 STOP
```

einfügen. Dadurch wird das Programm nach dem Anzeigen des Sprites unterbrochen. Im Gegensatz zu den Programmen mit der hochauflösenden Grafik ist das hier gefahrlos möglich. Wenn Sie das so geänderte Programm starten, bleibt das Sprite auf dem Bildschirm, und der Cursor erscheint wieder. In der dritten Bildschirmzeile steht "BREAK IN 60", und die Buchstaben werden teilweise vom Sprite verdeckt, sind also schlecht zu lesen. Dies ist ein Beleg für unsere Behauptung, daß Grafik/Text und Sprites gleichzeitig dargestellt werden können. Die Sprites scheinen dabei "vor dem Bildschirm" zu sein.

Sie können jetzt ganz normal weiter mit dem C64 arbeiten, das Sprite behindert das in keiner Weise (also auch kein SYNTAX ERROR), außer einigen schlecht lesbaren Zeichen auf dem Bildschirm.

Nun wollen wir uns aber einmal mit dem Aussehen des Sprites beschäftigen. Normalerweise müssen zur Darstellung des gewünschten Sprites 63 Speicherstellen durch POKE mit den richtigen Werten belegt werden. Wir haben aber für Sie eine viel

einfachere Weise herausgefunden, wie das Aussehen geändert werden kann. Geben Sie dazu bitte (bei eingeschaltetem Sprite auf dem Bildschirm) ein:

```
POKE 2040,16 (Return)
```

Wir verändern das erste Sprite

Sie werden sofort sehen, daß das Sprite sein Aussehen geändert hat, wahrscheinlich werden Sie drei senkrechte Streifen sehen. Nun, diese Änderung werden wir gleich noch erklären. Bewegen Sie den Cursor bitte in die oberste Zeile genau rechts neben das Sprite, und schreiben Sie folgenden Text (ohne Return): "Dies ist ein anderes Sprite". Achten Sie beim Schreiben auf das Aussehen des Sprites. Na, da kann man doch wirklich schön sehen, wie sich das Aussehen des Sprites ändert. Das funktioniert folgendermaßen:

Wir hatten ja schon darauf hingewiesen, daß für das Aussehen eines Sprites 63 Speicherstellen entscheidend sind. Diese Speicherstellen müssen aber nicht an einer bestimmten Stelle liegen. Vielmehr gibt eine Speicherstelle an, wo das Aussehen des Sprites abgelegt ist. Durch Änderung dieser Speicherstelle (POKE 2040,16) haben wir nun diesen Bereich genau an den Anfang des Bildschirmspeichers gelegt, also des 1000 Speicherstellen großen Bereiches, in dem die Zeichen für den Bildschirm liegen. Wenn wir diese Speicherstellen nun verändern (durch Eingabe von Zeichen), so ändern wir gleichzeitig den Bereich für das Sprite und damit sein Aussehen.

Wie entsteht ein Sprite?

Um Ihnen nun ein Gefühl dafür zu vermitteln, wie das Aussehen des Sprites durch diese Speicherstellen bestimmt wird, löschen Sie bitte den Bildschirm und bewegen den Cursor in die linke obere Ecke. Sie sehen, daß im selben Rhythmus wie der Cursor nun die linke obere Ecke des Sprites blinkt. Bewegen Sie den Cursor nun zweimal nach rechts. Jedesmal springt der blinkende Punkt des Sprites auch um ein Stück nach rechts. Wenn Sie den Cursor nun aber langsam weiterbewegen, sehen Sie, wie

der Punkt eine Zeile tiefer erneut beginnt und so fort. Das Aussehen des Sprites setzt sich also aus den Speicherstellen folgendermaßen zusammen:

```
   1  2  3      = Erste Zeile des Sprites.
   4  5  6      = Zweite Zeile des Sprites.
   7  8  9      = Dritte Zeile des Sprites.
...........
...........      (weitere Zeilen)
...........
  61 62 63      = Letzte Zeile des Sprites.
```

Nun kann aber jede Speicherstelle Werte zwischen 0 und 255 einnehmen und bestimmt jeweils 8 Punkte des Sprites. Dieser Zusammenhang ist nicht ganz einfach, und wir werden deshalb in unseren Programmen ein Unterprogramm benutzen, das uns die Arbeit des Umrechnens abnimmt. Damit Sie aber doch eine Vorstellung davon haben, wie das funktioniert, greifen wir auf ein wenig Mathematik zurück.

Die einzelnen 8 Punkte einer Speicherstelle erhält man dadurch, daß man Potenzen der Zahl zwei addiert. Für jeden zu setzenden Punkt muß die zugehörige Potenz der Zahl 2 addiert werden. Dabei wird von rechts nach links gezählt. Die 8 Punkte der Speicherstelle 001 entstehen also so:

```
2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0
```

Das spitze Dach entspricht dem Hochpfeil über der Return-Taste.

Um also den Punkt in der oberen linken Ecke des Sprites zu setzen, müssen wir in die Speicherstelle 001 eine 2^7=128 schreiben, und um die beiden obersten Punkte zu setzen, eine 192 (2^7+2^6=128+64=192) und so weiter. Jeder Punkt, der gesetzt werden soll, wird als Potenz von 2 dazu addiert, jeder nicht gesetzte Punkt entsprechend nicht. Das wollen wir gleich einmal an unserem Beispiel-Sprite ausprobieren.

Löschen Sie den Bildschirm, und bewegen Sie den Cursor in die 5. Zeile. Nun wollen wir die Speicherstelle 001 des Sprites gezielt verändern. Sie hat die Adresse 1024. Geben Sie also ein:

```
POKE 1024,2^7
```

Sie sehen, wie in der oberen linken Ecke des Bildschirms ein eigenartiges Zeichen erscheint (wir haben ja im Bildschirmspeicher die erste Speicherstelle verändert) und gleichzeitig auch die linke obere Ecke des Sprites anders aussieht. Der oberste Punkt des linken Balkens ist jetzt weiter nach links gerückt. Ändern Sie nun nacheinander in der POKE-Zeile die Werte von 7 bis 0. Jedesmal wandert der Punkt um eine Stelle weiter nach rechts. Bei 0 geht es natürlich jetzt nicht mehr weiter, denn negative Werte mag der C64 hier nicht. Wir haben ja auch nacheinander alle 8 Punkte der ersten Speicherstelle verändert, und nun wäre die zweite Speicherstelle 002 dran. Sie könnten wir genauso von links (= 7) nach rechts (= 0) verändern und so fort. Wichtig ist nur, daß nach jeweils drei Speicherstellen eine neue Zeile links begonnen wird.

Ein Sprite wird aufgebaut - Schritt für Schritt

Bevor wir nun mit diesen Kenntnissen ein wirklich schönes Sprite konstruieren, wollen wir noch ein geändertes Programm abdrucken, das nacheinander alle Punkte des Sprites setzt. Dadurch können Sie dann schön verfolgen, wie der Punkt nach 24 Schritten eine Zeile tiefer weitermacht. Sie müssen Zeile 65 einfügen und die Zeilen 100 - 200 anhängen.

```
10 REM "ERSTES SPRITE-PROGRAMM"
20 PRINT CHR$(147)
30 V = 53248
40 POKE V,200
50 POKE V+1,150
60 POKE V+21,1
65 GOSUB 100
70 GET A$: IF A$ = "" THEN 70
80 POKE V+21,0
90 END
100 POKE 1525,33
105 POKE 2040,16
110 FOR I = 1024 TO 1087
120 POKE I,0
130 NEXT I
140 FOR I = 1024 TO 1087
```

```
150 FOR J = 7 TO 0 STEP -1
160 POKE I,2^J
170 NEXT J
180 POKE I,255
190 NEXT I
200 RETURN
```

Nachdem das Sprite an einer geänderten Stelle des Bildschirms (40 POKE V,200 und 50 POKE V+1,150) eingeschaltet wurde und das Aussehen von den ersten Speicherstellen des Bildschirmspeichers abhängig ist, wird das Unterprogramm ab Zeile 100 abgearbeitet.

Zuerst wird neben die obere Ecke des Sprites ein Ausrufezeichen gesetzt, dann werden alle 63 Speicherstellen mit Nullen gefüllt, also ist kein Sprite-Punkt mehr sichtbar. Anschließend wird für alle 63 Speicherstellen nacheinander von links (7) nach rechts (0) die entsprechende Zweierpotenz hineingeschrieben und dadurch der zugehörige Punkt gesetzt.

Sind alle 8 Punkte der Speicherstelle fertig (J-Schleife beendet), wird eine 255 hineingeschrieben. Dadurch werden dann alle Punkte auf einen Schlag gesetzt. Nach der 63. Speicherstelle kehrt das Unterprogramm zum Hauptprogramm zurück.

9.3 Wir erstellen ein Sprite

Nachdem Sie nun eine Vorstellung davon haben, wie das Aussehen eines Sprites festgelegt wird, wollen wir endlich einmal ein professionelles Sprite erstellen. Sie können sich sicher vorstellen, wie schwierig es ist, die 63 Speicherstellen mit je 8 Punkten richtig aus den Zweierpotenzen zusammenzusetzen. Deshalb gehen wir einen einfacheren Weg.

Wir schreiben ein Unterprogramm, das das Aussehen des Sprites aus DATA-Zeilen liest und entsprechend zusammensetzt. Dabei wird in den DATA-Zeilen für jeden zu setzenden Punkt ein "*" geschrieben, für jeden Punkt, der nicht gesetzt werden soll, ein "-". Wie im Kapitel 8 bei dem Unterprogramm zum Setzen eines Punktes gilt auch hier: Sie müssen dieses Unterprogramm nicht

verstehen, denn Sie können es einfach in jedem Programm benutzen. Wollen Sie also ein neues Programm erstellen, das ein Sprite erzeugt, so laden Sie einfach das folgende Programm und bauen Ihr eigenes Programm darauf auf. Berücksichtigen Sie aber bitte, daß das Unterprogramm bestimmte Variablen benutzt und diese anschließend natürlich verändert sind. Am besten rufen Sie das Unterprogramm möglichst zu Anfang des Hauptprogramms auf.

```
10 REM "Sprite-Grundprogramm"
20 PRINT CHR$(147)
30 V = 53248
35 POKE 2040,13  :REM SPRITEDATEN LIEGEN AB 832
40 POKE V,50
50 POKE V+1,50
60 POKE V+21,1
65 GOSUB 40000
70 GET A$: IF A$ = "" THEN 70
80 POKE V+21,0
90 END
40000 REM UNTERPROGRAMM: SPRITE ERSTELLEN
40010 FOR I = 0 TO 62  : REM FUER ALLE 63 SPEICHERSTELLEN
40020 READ N$          : REM ZEICHENKETTE AUS DATAS HOLEN
40030 POKE 832+I,0     : REM SPEICHERSTELLE LOESCHEN
40040 FOR J = 1 TO 8   : REM FUER ALLE 8 PUNKTE
40050 IF MID$(N$,J,1)<>"*" THEN 40070
40060 POKE832+I,(2^(8-J))OR(PEEK(832+I))
40070 NEXT J
40080 NEXT I
40090 RETURN
40100 REM SPRITE IN DATA-ZEILEN
40101 DATA ********,********,********
40102 DATA *-------,--------,-------*
40103 DATA *-------,--------,-------*
40104 DATA *-------,--------,-------*
40105 DATA *-------,--------,-------*
40106 DATA *-------,--------,-------*
40107 DATA *-------,--------,-------*
40108 DATA *-------,--------,-------*
40109 DATA *-------,--------,-------*
40110 DATA *-------,--------,-------*
40111 DATA *-------,--------,-------*
40112 DATA *-------,--------,-------*
40113 DATA *-------,--------,-------*
40114 DATA *-------,--------,-------*
```

```
40115 DATA *-------,--------,-------*
40116 DATA *-------,--------,-------*
40117 DATA *-------,--------,-------*
40118 DATA *-------,--------,-------*
40119 DATA *-------,--------,-------*
40120 DATA *-------,--------,-------*
40121 DATA ********,********,********
```

Wenn Sie dieses Programm starten, wird ein Sprite auf den Bildschirm gezeichnet, das nur aus einem Rahmen besteht. Dadurch sehen Sie genau, wie groß das Sprite auf dem Bildschirm ist. In der Zeile 35 wird durch POKE übrigens festgelegt, daß die 63 Speicherstellen für das Aussehen des Sprites ab Speicherstelle 832 liegen. Speichern Sie dieses Grundprogramm unter dem Namen "Spritemaske" auf der Diskette ab, damit Sie es als Basis für weitere Programme verwenden können.

9.4 Der Ballon

Für unsere weiteren Experimente mit Sprites wollen wir natürlich nicht diesen "Kasten" verwenden. Statt dessen ändern wir das Aussehen des Sprites folgendermaßen ab:

```
40100 REM SPRITE IN DATA-ZEILEN
40101 DATA -------*,********,*-------
40102 DATA ------**,********,**------
40103 DATA -----***,********,***-----
40104 DATA -----***,***-****,***-----
40105 DATA ----****,********,****----
40106 DATA ----****,*-*--*-*,****----
40107 DATA ----****,**----**,****----
40108 DATA -----**-,*------*,-** ----
40109 DATA -----***,*------*,***-----
40110 DATA -----*-*,**----**,*-*-----
40111 DATA -----*-*,***--***,*-*-----
40112 DATA ------*-,*-****-*,-*------
40113 DATA ------*-,****-***,-*------
40114 DATA ------*-,-******-,-*------
40115 DATA ------*-,--****--,-*------
40116 DATA -------*,--****--,*-------
40117 DATA -------*,---**---,*-------
40118 DATA -------*,---**---,*-------
40119 DATA --------,*------*,--------
```

```
40120 DATA --------,********,--------
40121 DATA --------,********,--------
```

Laden Sie also das Programm "Spritemaske", ändern Sie die DATA-Zeilen entsprechend ab, und speichern Sie das geänderte Programm unter dem Namen "SpriteBallon" auf Diskette ab. Wenn Sie es anschließend starten, wird ein Ballon auf dem Bildschirm erscheinen. Diesen wollen wir jetzt für unsere weiteren Experimente verwenden. Vielleicht ist Ihnen schon aufgefallen, daß wir in allen bisherigen Sprite-Programmen zwei Zeilen ohne jede Erklärung benutzt haben:

```
40 POKE V,50
50 POKE V+1,50
```

Nun wollen wir einfach ausprobieren, was es mit den beiden Zeilen - beziehungsweise den hier verwendeten Speicherstellen - auf sich hat. Dazu wollen wir natürlich den Ballon auf dem Bildschirm haben, um mögliche Änderungen sofort feststellen zu können. Wir dürfen also das Sprite zum Schluß nicht in Zeile 80 ausschalten. Daher fügen wir eine Zeile ein:

```
75 STOP
```

Wenn Sie das Programm nun starten und der Ballon fertig ist, bricht das Programm nach dem Drücken einer Taste ab, der Ballon bleibt sichtbar. Geben Sie nun einmal folgende Zeile ein:

```
POKE V,250: POKE V+1,150
```

Nach dem Drücken der Eingabetaste macht unser Ballon einen gewagten Sprung nach rechts unten. Diese beiden Speicherstellen haben also anscheinend etwas mit der Position des Ballons auf dem Bildschirm zu tun. Genauer steht in V die X-Position und in V+1 die Y-Position. Indem wir in diese beiden Speicherstellen neue Werte hineinschreiben, können wir den Ballon sofort auf eine andere Stelle des Bildschirms setzen.

9.5 Sprites bewegen

Nun wird ein echter Ballon natürlich nicht mit einem gewaltigen Satz seine Position ändern, sondern sich langsam bewegen. Dies können wir aber ganz einfach erreichen, indem wir in einer Schleife die Position des Ballons nur um 1 ändern und dann noch eine kleine Verzögerung durch eine Warteschleife veranlassen. Geben Sie deshalb folgende Zeile ein:

```
FOR I=50 TO 226:FOR J=1 TO 20:NEXT J: POKE V+1,I: NEXT I
```

Und schon beginnt unser Ballon, langsam vom Himmel herunterzuschweben, und landet genau am unteren Rand. So, wie wir den Ballon über den Bildschirm gewegt haben, passiert das auch in professionellen Spielen, nur noch schneller und mit einigen zusätzlichen Tricks. Trotzdem können wir uns jetzt erst einmal auf unserem Erfolg ausruhen und die Früchte (Ballons) genießen. Wenn Sie während dieser Entspannung übrigens den Ballon über den Bildschirm fliegen - Pardon: fahren - lassen wollen, dann fügen Sie in das Programm "SpriteBallon" einfach folgende Zeilen ein:

```
70 GET A$: IF A$ <> "" THEN 80
75 FOR I = 1 TO 228: FOR J = 1 TO 20: NEXT J
77 POKE V+1,I: NEXT J: GOTO 70
```

Nun fährt der Ballon ständig über den Bildschirm. Wenn er am unteren Rand angekommen ist, beginnt er wieder oben. Während wir uns nun gemeinsam entspannen, wollen wir Ihnen noch einige Hintergründe zu Sprites erklären. Mit Sprites können Sie nämlich wirklich fantastische Möglichkeiten aus Ihrem C64 herausholen:

Sprites sind keine Singles: Acht auf einen Streich

Beginnen wir damit, daß der C64 nicht nur ein einziges, sondern bis zu 8 Sprites gleichzeitig erzeugen kann. Diese können völlig unterschiedlich aussehen und sich unterschiedlich bewegen. Dementsprechend mehr Daten müssen wir natürlich dem C64 mitteilen. Es gibt aber eine ganz einfache Möglichkeit, mehrere Sprites auf dem Bildschirm erscheinen zu lassen, wenn diese

gleich aussehen. Dazu sagen wir dem VIC (dem Spezialbaustein für die Grafik) einfach, daß er das Aussehen für ein weiteres Sprite aus demselben Bereich holen soll wie das Aussehen für das erste. Außerdem müssen wir dieses zweite Sprite natürlich einschalten, seine Position festlegen und es bewegen. Ändern Sie im bisherigen Programm einfach folgende Zeilen ab:

```
30 POKE V,50: POKE V+2,100
40 POKE V+1,50: POKE V+3,50
50 POKE 2040,13: POKE 2041,13
60 POKE V+21,3

77 POKE V+1,I: POKE V+3,I: NEXT J: GOTO 70
```

Wenn Sie diese 5 Zeilen entsprechend geändert haben, bewegen sich 2 gleiche Ballons gleichmäßig über den Bildschirm.

Sprites mögen (und vertragen) Farbe!

Vielleicht fällt Ihnen bei den Sprites doch ein Unterschied auf. Bei uns haben sie nämlich nicht die gleiche Farbe beziehungsweise die gleiche Graustufe. Der zweite Ballon ist viel dunkler als der erste. Der C64 kann nämlich jedem Sprite seine eigene Farbe geben. Diese befindet sich als Zahlenwert für Sprite 1 in Speicherstelle V+39, für Sprite 2 in V+40. Wenn die Ballons gerade schön sichtbar auf dem Bildschirm sind, dann unterbrechen Sie doch einfach das Programm mit RUN/STOP und geben ein:

```
POKE V+39,1: POKE V+40,1
```

Dann sind beide Ballons in der gleichen Farbe auf dem Bildschirm. Natürlich können Sie auch einfach andere Werte ausprobieren. Dabei sind alle Zahlen von 0 - 15 interessant, denn 16 verschiedene Farben kann der C64 darstellen.

Sprites - fast grenzenlos!

Vielleicht ist Ihnen auch schon aufgefallen, daß die Ballons aus dem oberen, nicht sichtbaren Teil des Bildschirms herunterschweben. Das liegt daran, daß der VIC die Sprites auch an Positionen außerhalb des sichtbaren Bildschirms verwalten kann,

nur sind sie dann eben nicht mehr sichtbar. Deshalb sind wir bei den meisten Beispielen auch immer mit Y-Position = 50 gestartet und haben bei Y = 228 die Bewegung nach unten beendet. Bei X = 20 und Y = 50 befindet sich der Ballon gerade noch ganz sichtbar in der oberen linken Ecke.

Bei der X-Position sieht es noch anders aus. Bei X = 0 ist der Ballon links noch nicht sichtbar, bei X = 255 (dem höchsten erlaubten Wert) dagegen hat er noch gar nicht den rechten Rand erreicht. Das ist ja auch einleuchtend, denn der Bildschirm besteht ja (siehe Kapitel 8) in der Breite aus 320 Punkten. Zwar gibt es auch eine Möglichkeit, den Ballon bis zum rechten Rand des Bildschirms zu bewegen, aber die ist so kompliziert und umständlich, daß wir an dieser Stelle darauf verzichten möchten.

Sprites im Crashtest

Eine ganz wichtige Eigenschaft haben Sprites noch, auf die wir unbedingt hinweisen möchten. Sie können nämlich mit etwas zusammenstoßen. Das ist für viele Anwendungen von Sprites äußerst wichtig und wird auch praktisch in jeder professionellen Anwendung und jedem Spiel genutzt. Das, womit ein Sprite zusammenstoßen kann, kann sowohl ein anderes Sprite als auch ein sichtbares Zeichen auf dem Bildschirm sein. Jetzt dürfen Sie natürlich nicht hoffen, daß der VIC die beiden Sprites beim Zusammenstoß "knautscht". Er gibt uns vielmehr nur ein Zeichen in einer bestimmten Speicherstelle, daß Sprites zusammengestoßen sind, und in einer anderen Speicherstelle, daß ein Sprite mit dem Hintergrund zusammenstieß.

Sicherlich haben Sie auf den letzten Seiten so richtig Gefallen an den Sprites gefunden und möchten nun die vorgestellten Möglichkeiten auch ausprobieren. Natürlich können wir Ihnen nicht alles in einem Einsteigerbuch zeigen. Also sind wir einen Mittelweg gegangen:

Zum einen stellen wir Ihnen im Anschluß noch ein etwas größeres Programm zum Einsatz von Sprites vor, das Ihnen hoffentlich genauso viel Spaß macht wie uns und viele der genannten Möglichkeiten auch auf einfache Weise nutzt.

Zum anderen möchten wir Sie auf ein Buch verweisen, das jede Frage zum Thema Grafik (und damit auch Sprites) umfassend beantwortet und Ihnen darüber hinaus noch eine ausgezeichnete Hilfestellung in Form von Hilfsprogrammen liefert - besser als unsere kleinen Unterprogramme jemals sein könnten: "Das Supergrafik-Buch zum C64". Es ist, wie der Name schon sagt, einfach super.

9.6 Das Ballonspiel

Bevor wir uns nun von den Sprites verabschieden, wollen wir Ihnen - wie versprochen - zeigen, wie man in BASIC ein Spiel mit Sprites programmiert. In diesem Spiel geht es darum, einen vom Himmel herunterschwebenden Ballon an der richtigen Stelle zu landen. Dazu kann er mit den Cursor-Tasten gesteuert werden. Im Gegensatz zu einem echten Ballon können wir ihn allerdings auch nach links und rechts steuern.

Nun wäre es natürlich einfach, den Ballon mit den Cursor-Tasten auf die gewünschte Stelle zu bewegen und herunterschweben zu lassen. Deshalb gibt es in unserem Spiel wechselnden Wind, der die Landung ganz schön schwierig macht. Aber am besten, Sie probieren das folgende BASIC-Programm einfach aus:

```
10 REM "    BALLON LANDEN      "
20 GOSUB  70:REM "INITIALISIERUNG
30 GOSUB  40000:REM "BALLON ERSTELLEN "
40 GOSUB  300:REM "SPIEL"
50 GOSUB 490:REM "ENDE"
60 END
70 REM "INITIALISIERUNG"
80 V = 53248
90 PRINTCHR$(147)
100 POKE V+21,0 :REM SPRITE 0 AUS
110 POKE 2040,13:REM SPRITE 0 AB 832
120 POKE V+39,1 :REM FARBE 1
130 POKE V+23,0 :REM Y-NORMAL
140 POKE V+29,0 :REM X-NORMAL
150 POKE V+30,0 :REM SPRITE-SPRITE AUS
160 POKE V+31,0 :REM SPRITE-ZEICHEN AUS
```

```
170 GX = 0
180 GY = 1
190 PX = 170
200 PY = 50
210 WX = 0
220 WY = 0
230 FOR I=1984 TO 2023
240 POKE I,119
250 NEXT I
260 FOR I = 2002 TO 2005
270 POKE I,32
280 NEXT I
290 RETURN
300 REM ********** SPIEL ***********
310 POKE V,PX
320 POKE V+1,PY
330 POKE V+21,1
340 FOR J = 1 TO 20: NEXT J
350 PX=PX+GX
360 PY=PY+GY
370 IF PX<0   THEN PX=PX+256
380 IF PX>255 THEN PX=PX-256
390 IF PY<0   THEN PY=PY+256
400 IF PY>255 THEN PY=PY-256
410 IF PY>228 AND PEEK(V+31)= 0 THEN RETURN
420 POKE V,PX
430 POKE V+1,PY
440 IF PEEK(V+31)>0 THEN POKEV+31,0:GOTO 800: REM BRUCHLANDUNG
450 GOSUB 630: REM TASTATUR
460 GY=GY+0.05
470 GOSUB 710: REM WIND
480 GOTO 340
490 REM ********** ENDE ***********
500 POKEV+0,150
510 POKEV+1,130
520 POKEV+23,1
530 POKEV+29,1
550 PRINTCHR$(147)
560 PRINT:PRINT:PRINT:PRINT
570 PRINT"            G E L A N D E T ! "
580 PRINT"            ----------------- "
590 GET A$:IFA$="" THEN 550
600 POKE V+21,0
610 PRINTCHR$(147)
620 RETURN
630 REM *** TASTATUR ABFRAGEN ***
```

```
640 GET A$: IF A$ = "" THEN RETURN
650 A = ASC(A$)
660 IF A= 17 THEN GY=GY+1
670 IF A= 29 THEN GX=GX+1
680 IF A=157 THEN GX=GX-1
690 IF A=145 THEN GY=GY-1
700 RETURN
710 REM ********** WIND ***********
720 IF RND(1)<0.1THEN WX=WX+0.1:GOTO770
730 IF RND(1)<0.1THEN WY=WY+0.1:GOTO770
740 IF RND(1)<0.1THEN WX=WX-0.1:GOTO770
750 IF RND(1)<0.1THEN WY=WY-0.1:GOTO770
760 IF RND(1)<0.1THEN WX=0:WY=0 :GOTO770
770 GX=GX+WX
780 GY=GY+WY
790 RETURN
800 REM ***** BRUCHLANDUNG *******
810 FOR I = 0 TO 15
820 POKE V+39,I
830 FOR J=1 TO 10: POKE53280,J: NEXT J
840 NEXT I
850 POKE V+39,1
860 FOR J=1 TO 1000:NEXT J
870 POKEV+21,0
880 PRINT CHR$(147)
890 END
40000 REM **** SPRITE ERSTELLEN ******
40010 FOR I = 0 TO 62
40020 READ N$
40030 POKE832+I,0
40040 FOR J = 1 TO 8
40050 IF MID$(N$,J,1) <> "*" THEN 40070
40060 POKE832+I,(2^(8-J))OR (PEEK(832+I))
40070 NEXT J
40080 NEXT I
40090 RETURN
40100 REM **** SPRITE IN DATAS ******
40110 DATA -------*,********,*-------
40120 DATA ------**,********,**------
40130 DATA -----***,********,***-----
40140 DATA -----***,***-****,***-----
40150 DATA ----****,********,****----
40160 DATA ----****,*-*--*-*,****----
40170 DATA ----****,**----**,****----
40180 DATA -----**-,*------*,-**-----
40190 DATA -----***,*------*,***-----
```

```
40200 DATA -----*-*,**----**,*-*-----
40210 DATA -----*-*,***--***,*-*-----
40220 DATA -----*--,*-****-*,--*-----
40230 DATA ------*-,****-***,-*------
40240 DATA ------*-,-******-,-*------
40250 DATA ------*-,--****--,-*------
40260 DATA -------*,--****--,*-------
40270 DATA -------*,---**---,*-------
40280 DATA -------*,---**---,*-------
40290 DATA --------,*------*,--------
40300 DATA --------,********,--------
40310 DATA --------,********,--------
```

Wenn Sie diese BASIC-Zeilen richtig eingegeben haben, speichern Sie das Spiel bitte unter dem Namen "Ballonspiel" auf der Diskette ab. Wenn Sie es mit RUN starten, dauert es zuerst einige Sekunden, bis das Sprite erstellt ist und das Spiel mit dem Ballon am oberen Bildschirmrand beginnt. Wenn Sie das Spiel noch weiter verfeinern oder ausbauen möchten, finden Sie im folgenden eine Erklärung der wichtigsten Programmzeilen:

Die Zeilen 10 - 60 stellen das Hauptprogramm dar, das die einzelnen Unterprogramme aufruft. Ab Zeile 70 beginnt die Initialisierung, in der wichtige Variablen auf den richtigen Wert gesetzt werden. V ist die Adresse des VIC. Zuerst wird in Zeile 100 unser Sprite ausgeschaltet, damit man nicht sehen kann, wie es Punkt für Punkt aufgebaut wird. In der nächsten Zeile werden die Daten für das Aussehen des Sprites festgelegt. Sie liegen ab Speicherstelle 832. Später werden wir die Werte aus den DATA-Zeilen dort hineinschreiben. Mit POKE V+31,1 wird die Farbe des Sprites auf "weiß" gesetzt. Die Zeilen 130 und 140 haben entscheidenden Einfluß auf die Größe des Sprites. Würden wir in beide Speicherstellen eine "1" schreiben, dann wäre das Sprite in X- und Y-Richtung doppelt so groß. Dabei wird jeder Punkt in der jeweiligen Richtung einfach verdoppelt. Angenommen, ein Ausschnitt aus unserem Sprite sähe so aus:

```
...****...
..******..
.********.
```

Wenn wir dann mit POKE V+29,1 in Y-Richtung verdoppeln würden, erschiene das Sprite so auf dem Bildschirm:

```
......********......
....************....
..****************..
```

Wir schalten aber in unserem Fall die Vergrößerung erst einmal auf Null, damit der Ballon überhaupt landen kann. In den Zeilen 150 und 160 werden die Speicherstellen zur Kollisionsprüfung auf 0 gesetzt. Sollte vorher schon einmal eine Kollision erfolgt sein, würde unser Programm gleich nach dem Start meinen, das Spiel wäre beendet.

Die Programmzeilen ab 170 legen Position (PX,PY), Geschwindigkeit (GX,GY) und Wind (WX,WY) für das Spiel fest. Die Zeilen 230-250 sorgen für die untere Begrenzung des Spielfeldes. Die folgende Schleife erzeugt durch die Leerzeichen eine Lücke in diesem Boden, durch die später der Ballon gelandet werden muß. In Zeile 290 endet das Unterprogramm mit RETURN.

Das eigentliche Spiel läuft im Unterprogramm ab Zeile 300 ab. Zuerst wird die Position des Sprites (PX,PY) in die richtigen Speicherstellen geschrieben. Dann wird das Sprite eingeschaltet. Der folgende Teil ab Zeile 340 wird solange wiederholt, bis das Spiel beendet ist. Zuerst erfolgt eine kleine Verzögerung durch die Schleife bis 20. Ohne diese Verzögerung würde das Spiel zu schnell ablaufen. In den folgenden beiden Zeilen wird die neue Position errechnet, und zwar aus der alten Position plus Geschwindigkeit. Ist also die Geschwindigkeit GX "1", so wird bei jedem Durchlauf der Ballon um einen Punkt nach rechts bewegt. Die neue Position darf nun aber nicht einfach benutzt werden, vielmehr muß sie erst auf den erlaubten Bereich hin überprüft werden. Sie darf ja nicht kleiner als 0 oder größer als 255 sein. Die vier Programmzeilen 370-400 sorgen dafür, daß PX und PY immer zwischen 0 und 255 liegen.

In Zeile 410 wird geprüft, ob der Ballon schon gelandet ist. Dann ist seine Y-Position nämlich größer als 228. Gleichzeitig darf aber keine Kollision mit der unteren Linie stattgefunden haben (PEEK(V+31) muß = 0 sein). In diesem Fall endet das Unterprogramm "Spiel", und das Hauptprogramm kann fortfahren. In den Zeilen 420-430 wird die auf erlaubten Bereich geprüfte neue Position des Sprites in die entsprechenden Speicher-

stellen geschrieben. Dadurch wird der Ballon auf dem Bildschirm bewegt.

Zeile 440 prüft, ob der Ballon mit der unteren Linie zusammengestoßen ist. Dann steht in der Speicherstelle V+31 nämlich eine 1. Diese Speicherstelle wird wieder ordnungsgemäß auf 0 gesetzt, und ab Zeile 800 wird ein Programmteil aufgerufen, der die "Bruchlandung" anzeigt und das Spiel beendet. Zeile 450 ruft ein Unterprogramm auf, das die Tastatur abfragt und entsprechend die Geschwindigkeit verändert. Da ein Ballon nicht einfach in der Luft schwebt, sondern langsam sinkt, wird in Zeile 460 die Y-Geschwindigkeit immer etwas erhöht.

Den gesamten Einfluß des Windes erledigt das Unterprogramm ab Zeile 710. In Zeile 480 ist unsere Spielschleife beendet und beginnt erneut ab Zeile 340.

Das Unterprogramm "Ende" ab Zeile 490 wird vom Hauptprogramm aufgerufen, wenn der Ballon richtig gelandet wurde. Es setzt den Ballon mitten auf den Bildschirm und verdoppelt seine Größe in beiden Richtungen (V+23, V+29). Die Zeilen 550-590 werden solange durchlaufen, bis eine Taste gedrückt wird. Zuerst wird der Bildschirm gelöscht, dann wird der Cursor in die 6. Zeile bewegt und die Erfolgsmeldung mitten auf den Bildschirm geschrieben. Wurde noch keine Taste gedrückt, so beginnt der Vorgang erneut. Durch das wechselnde Löschen und Ausgeben der Meldung scheint diese zu blinken. Nach dem Druck einer Taste wird das Sprite ausgeschaltet, der Bildschirm gelöscht und das Unterprogramm beendet.

Ab Zeile 630 beginnt die Tastaturabfrage. Wurde keine Taste gedrückt, so ist das Unterprogramm in Zeile 640 sofort beendet. Ansonsten wird der ASCII-Wert des Zeichens in A$ bestimmt und in den Zeilen 660 bis 690 mit den Cursor-Tasten verglichen. Für jede dieser Tasten wird die Geschwindigkeitskomponente um 1 geändert, also beispielsweise bei Cursor-Rechts GX um "1" erhöht.

Das Unterprogramm zur Erzeugung des Windes beginnt in Zeile 710. Für jede der vier Richtungen wird durch Zufall bestimmt,

ob sich der Wind ändert. Da RND(1) eine Zahl zwischen 0 und 1 liefert, tritt der Fall " < 0.1 " im Schnitt nur jedes zehnte Mal auf, und auch dann wird der Wind nur um 0.1 geändert. Das scheint zwar sehr wenig zu sein, aber wenn Sie hier probeweise höhere Werte einsetzen, werden Sie schnell feststellen, daß Sie schon ein echter Profi-Balloner sein müssen, um noch sicher zu landen. Das liegt daran, daß der Wind ja bei jedem Durchlauf zur Geschwindigkeit addiert wird, und das gibt schnell große Werte.

Wenn eine der Bedingungen für eine Windänderung eintritt, werden die anderen Bedingungen nicht mehr getestet (GOTO 770), damit die gerade erfolgte Änderung wieder aufgehoben wird (+0.1 und -0.1). Ab Zeile 770 wird der Wind zur Geschwindigkeit addiert und wirkt sich damit voll auf den Ballon aus.

Ab Zeile 800 steht der Programmteil, der bei einer Bruchlandung (Berühren des unteren Striches) aufgerufen wird. Wir haben hier zwei ineinander geschachtelte Schleifen eingesetzt. In der äußeren I-Schleife wird der I-Wert als Farbwert für den Ballon benutzt (POKE V+39,I). Anschließend wird in der inneren J-Schleife die Rahmenfarbe des Bildschirms geändert. Die innere Schleife wird dabei 10*15 = 150mal durchlaufen, so daß der Bildschirmrand ganz schön blinkt. Nach diesen effektvollen Schleifen wird die Farbe des Sprites wieder auf "1 = weiß" gesetzt, und nach der Verzögerungsschleife in Zeile 860 wird der Ballon dann endgültig vom Bildschirm gelöscht (POKE V+21,0). Die Programmzeilen ab Zeile 40000 zum Erstellen des Sprites kennen Sie ja nun schon.

So, das wäre geschafft. Jetzt können Sie sich dem Spiel in dem süßen Gefühl hingeben, es auch verstanden und ggf. auch ein wenig manipulieren zu können. Zum Beispiel kann der Wind ja ein wenig heftiger oder auch ein wenig lascher wehen, je nachdem, wer gerade spielt.

9.7 Zusammenfassung

1. Sprites stellen neben ASCII-Zeichen-Grafik und der hochauflösenden Grafik eine weitere und besondere Darstellungsmöglichkeit des C64 dar, die ihn berühmt gemacht und zu seinem großen Erfolg beigetragen hat.

   Sprites können 24 Punkte breit und 21 Punkte hoch sein. Da in der Breite immer 8 Punkte in einen Speicherplatz passen, werden 63 Speicherplätze benötigt. Sie befinden sich in einem speziellen Speicherbereich. Dadurch können Sprites im Textmodus nicht nur auf dem Bildschirm stehenbleiben, sondern sogar durch einfaches Überschreiben nicht gelöscht werden. Sie scheinen vor dem normalen Bildschirm zu liegen.

2. "Angestellt" wird der Sprite-Modus durch die folgenden Programmzeilen:

   ```
   30 V = 53248
   40 POKE V,50
   50 POKE V+1,50
   60 POKE V+21,1
   ```

   Ausgestellt wird er durch:

   ```
   80 POKE V+21,0
   ```

3. Die genaue Form, das Aussehen eines Sprites wird auf recht komplizierte Weise bestimmt, die Sie in entsprechender, weiterführender Literatur (beispielsweise dem "Supergrafik-Buch" von DATA BECKER) erklärt bekommen. Wir empfehlen Ihnen die Benutzung eines einfachen und immer wieder verwendbaren Unterprogramms, das wir Ihnen abgedruckt haben. In ihm brauchen Sie nur einige DATA-Zeilen (jeweils natürlich mit RETURN abgeschlossen) zu ändern. Dann sehen Sie bereits im Programm das ungefähre Aussehen des Sprites. Und anschließend haben Sie es auch wirklich auf dem Bildschirm.

4. Bis zu acht Sprites können verwaltet werden. Dabei muß man nur darauf achten, daß ihr Aussehen an der richtigen Stelle abgelegt und dies dem VIC ab Adresse 53248 mit-

geteilt wird. Wenn Sprites dasselbe Aussehen haben, kann auf dieselben Speicherstellen zurückgegriffen werden.

5. Man kann Sprites mit einer gewünschten Farbe versehen. Dies geschieht, indem man in V+39 eine Zahl zwischen 0 und 15 schreibt, also beispielsweise

   ```
   POKE V+39,1
   ```

   für ein weißes, erstes Sprite. In die folgenden 7 Speicherstellen V+40, V+41 usw. kommen dann die Farben eventueller weiterer Sprites.

6. Sprites können in der Vertikalen auch den Bildschirm verlassen. In der Horizontalen allerdings gibt es eine rechte Grenze, die nur recht kompliziert überschritten werden kann. Sie können nicht einmal so ohne weiteres den normalen Bildschirmrand erreichen.

7. Eine besondere Möglichkeit stellt das Zusammentreffen von Sprites, ihre Kollision dar, für viele Spiele eine unverzichtbare Grundlage. Dabei wird in Speicherstelle V+30 ein Signal vom VIC gesetzt, dort erscheint dann eine Zahl ungleich 0, die Sie mit PEEK(V+30) erfragen können. Ebenso kann in V+31 ein Zusammenstoß eines Sprite mit einem sichtbaren Teil des Bildschirms geprüft werden.

10. Ausblicke für Aufsteiger

Sicherlich hat Ihnen die Beschäftigung mit BASIC (und damit auch das Kennenlernen Ihres C64) sehr viel Spaß gemacht. Wir sind nun gemeinsam an dem Punkt angekommen, an dem Sie durch eigene Programme und Ausprobieren Ihre neuerworbenen Fähigkeiten weiter ausbauen können. In diesem Kapitel möchten wir Ihnen noch einige Ausblicke geben, wie es nun weitergehen könnte. Besonders wichtig ist uns auch, Ihnen dabei noch ein paar Tips zu geben, wie Sie bei möglicherweise auftauchenden Problemen vorgehen können und wo Sie Informationen und Hilfsmittel erhalten. Noch ein kurzer Hinweis vorweg: Wir standen natürlich vor der Wahl, eher Allgemeingültiges von uns zu geben oder aber ganz konkrete Hilfen anzubieten. Wir haben uns verständlicherweise ganz im Sinne dieses Buches für die letzte Möglichkeit entschieden. Das bedeutet aber natürlich, daß alle ganz konkreten Informationen, etwa der Titel oder die Lieferbarkeit eines Buches, sich auf den Zeitpunkt der Veröffentlichung dieses Buches beziehen. Aber diese kleine Einschränkung nehmen Sie sicherlich gerne in Kauf, wenn Sie dafür unmittelbar an unseren Erfahrungen teilhaben können.

10.1 Weiterführende Literatur

Wie Sie hoffentlich an diesem Buch gesehen haben, erschließt sich die Wunderwelt der Computer sehr viel leichter, wenn man entsprechend gut gemachte Bücher als hilfreiche Begleiter zur Verfügung hat. Was für die kurze Zeit des Einsteiger-Daseins gilt, gilt auch für die Zeit danach, wenn Sie sich immer tiefer in die Möglichkeiten Ihres C64 einarbeiten. Aus diesem Grunde möchten wir Ihnen zunächst einige weitere Bücher präsentieren. Sie sind im selben Verlag erschienen wie das vorliegende Buch. Natürlich gibt es auch andere Verlage und andere gute Bücher. Wir beschränken uns hier aber auf Angebote aus dem Verlag DATA BECKER, weil wir mit diesen Büchern sehr viel arbeiten und sie deshalb auch am besten beurteilen können. Und es geht ja um unsere guten Erfahrungen, die wir weitergeben wollen.

Das große Floppybuch zur 1541

Haben Sie an der Datenspeicherung auf Diskette Spaß gefunden und wollen Sie diese Methode in eigenen Programmen einsetzen? Sind Sie an weiteren interessanten Möglichkeiten der Floppy interessiert, und hat Sie ein Blick ins mitgelieferte Handbuch bisher eher abgeschreckt? Dann sollten Sie sich unbedingt "Das große Floppybuch zur 1541" von DATA BECKER zulegen. Es beginnt beim Laden und Speichern, zeigt neben dem Löschen von Dateien und Formatieren einer Diskette alle weiteren interessanten Befehle im Umgang mit der Floppy und erklärt natürlich ausführlich den Umgang mit den (schon bekannten) sequentiellen und den relativen Dateien. Daß es dabei noch viele interessante Möglichkeiten gibt, brauchen wir Ihnen sicherlich nicht zu verraten. Daß dieses Buch Sie bis zum professionellen Umgang mit der Floppy sicher begleitet, macht es zu einer dauerhaften Anschaffung.

Das Supergrafik-Buch zum C64

Wir haben ja schon bei der Programmierung der hochauflösenden Grafik und der Sprites darauf hingewiesen, daß wir Ihnen leider nicht alle Möglichkeiten zeigen, aber ein Buch wärmstens empfehlen können: "Das Supergrafik-Buch zum C64". Es enthält nicht nur alle Informationen zum Thema Grafik mit ausführlichen Beispielen und vielen Tips, sondern im Preis ist auch eine randvolle Diskette enthalten. Auf dieser befindet sich das Programm "Supergrafik", das den Namen mehr als verdient. Es unterstützt Sie in einer derart perfekten Weise bei der Programmierung des C64 und seiner grafischen Möglichkeiten, daß Sie unsere kleinen Unterprogramme bald beiseite legen werden. Mit den neuen (BASIC-)Befehlen, die Ihnen die Supergrafik zur Verfügung stellt, erstellen Sie Kreise, Balken, zeichnen Sie Rechtecke, füllen Sie Flächen ... schon das Inhaltsverzeichnis des Buches ist 5 Seiten lang. Und alle Programme befinden sich auch auf der Diskette, so daß Sie sich das fehlerträchtige Abtippen sparen können: Ein heißer Tip nicht nur für lange Winterabende.

C64 Tips und Tricks

Erinnern Sie sich noch an den Trick im Sprite-Kapitel, wo wir die Speicherstellen für das Aussehen des Sprites so gelegt haben, daß der Cursor im oberen linken Bildschirmbereich das Aussehen des Sprites ändern konnte. Gerade bei der eigenen Programmierung braucht man Jahre, um alle wichtigen Tips und Tricks zu kennen und damit anderen voraus zu sein - oder "C64 Tips und Tricks" von DATA BECKER. Dieses Buch ist nicht ohne Grund über 100000mal verkauft worden. Auf 418 Seiten finden Sie zum Beispiel einen Trick, den Cursor viel schneller auf eine bestimmte Stelle des Bildschirms zu bewegen als mit unserem (langsamen) Unterprogramm. Daß dadurch die Balken "in den Himmel schießen", dürfte klar sein. Oder möchten Sie das Inhaltsverzeichnis der Diskette anschauen, ohne jedesmal das Programm im Speicher zu verlieren? Kein Problem mit dem entsprechenden Trick aus diesem Buch.

Maschinensprache für Einsteiger

Ein Buch möchten wir Ihnen noch als Geheimtip empfehlen, obwohl es Sie vielleicht im ersten Augenblick wundern wird, daß die Autoren eines "BASIC für Einsteiger" Ihnen ein solches Buch empfehlen: "Maschinensprache für Einsteiger". Der Grund ist einfach. Gerade auf dem C64 wird Maschinensprache sehr oft zusätzlich zum BASIC eingesetzt, weil dadurch vieles einfacher und schneller wird. Deshalb werden Sie auch in vielen Büchern und Zeitschriften immer wieder innerhalb von BASIC-Programmen auf Maschinensprache stoßen - oft als DATA-Zeilen. Dieses Buch führt Sie in erfreulich kleinen Schritten in die Welt der Maschinensprache ein und zeigt Ihnen, wo und wie man diese neuen Möglichkeiten richtig einsetzt.

10.2 Weitere Möglichkeiten von BASIC

Natürlich konnten wir Ihnen in diesem Einsteigerbuch nicht alle Möglichkeiten und Befehle von BASIC zeigen. So gibt es nicht nur weitere Befehle, sondern durch die Kombination dieser Befehle lassen sich noch viele neue Möglichkeiten verwirklichen.

Aber BASIC ist nicht gleich BASIC. So unterscheiden sich beispielsweise die BASIC-Versionen der verschiedenen Computerfirmen ganz erheblich voneinander. Und beim C64 ist man leider in puncto BASIC sehr sparsam gewesen. Daher wurden gerade für den C64 eine Fülle neuer BASIC-Versionen programmiert, die viel leistungsfähiger sind als das Original und - und das dürfte für Sie am Anfang viel wichtiger sein - viel komfortabler.

An dieser Stelle können wir Ihnen natürlich nicht alle, noch nicht einmal die wichtigsten vorstellen. Aber wenn Sie auch einmal ein Programm mit neuen Zeilennummern versehen wollen, 30 Programmzeilen mit einem Befehl löschen, die wichtigsten Befehle auf den Funktionstasten (F1 - F8) und damit mit einem Tastendruck benutzen wollen, ein Programm Schritt für Schritt abarbeiten lassen wollen, um einen Fehler leicht zu finden, oder oder oder. All das bieten sogenannte BASIC-Erweiterungen, und sie können einem das Programmieren schon ganz schön erleichtern.

Ein besonders leistungsfähiger Vertreter dieser BASIC-Erweiterungen ist BECKERbasic 64 aus dem Hause DATA BECKER. Von dieser Erweiterung muß man wohl sagen, daß sie in jeder Hinsicht Superlative bietet: 273! neue, leistungsstarke Befehle (zum Vergleich: das Original-C64-BASIC bietet insgesamt ca. 60), Hilfen und Vereinfachungen für alle Bereiche der Programmierung: von Grafik und Sprites über Floppy bis zur Veränderung von Programmzeilen. Die sensationellste Möglichkeit wird aber wohl alle Benutzer der grafischen Benutzeroberfläche GEOS begeistern: BECKERbasic 64-Programme laufen unter GEOS und können viele der Möglichkeiten von GEOS nutzen. Ach ja, bleibt noch der Preis. Über den braucht man unserer Meinung nach wirklich nicht zu diskutieren, er ist entwaffnend günstig.

10.3 So wird BASIC schneller - Compiler

Gerade bei der Benutzung von Grafik haben Sie sicherlich auch gemerkt, daß BASIC, was die Geschwindigkeit angeht, seine Grenzen hat. Das Löschen der hochauflösenden Grafik dauert um die 45 Sekunden, und auch beim Zeichnen der Balken kann man gemütlich zuschauen. Das haben natürlich auch schon viele andere Programmierer zu ihrem Leidwesen festgestellt und Abhilfe geschaffen: Compiler.

BASIC ist unter anderem deshalb so langsam, weil der C64 jede Zeile des Programms vor der Ausführung genau prüft. SYNTAX ERROR und UNDEF'D STATEMENT ERROR werden dabei festgestellt, und das Programm wird beendet. Besonders nachteilig ist das in Programmteilen, die viele Male abgearbeitet werden, denn leider erfolgt die Prüfung hier auch viele Male, also bei jedem Programmablauf. So wird also in einer hundertmal durchlaufenen Schleife dieselbe Prüfung auch hundertmal vorgenommen. Ein Compiler dagegen prüft das Programm vor dem Programmablauf und macht damit den späteren vielfachen Prüfungsprozeß überflüssig und das Programm somit schneller.

Natürlich ist ein so verändertes Programm nicht mehr in BASIC zu verändern. Deshalb speichert man die normale Version und die veränderte, kompilierte einzeln ab und hat damit eine langsame Version zum nachträglichen Ändern und eine schnelle für die Ausführung.

Zusätzlich machen die Compiler üblicherweise etwas, was ihnen auch ihren Namen gegeben hat: Sie übersetzen das Programm. Besonders gute Compiler übersetzen das Programm in reine Maschinensprache, die das Programm um ein Vielfaches schneller macht. Wenn Sie sich für einen solchen Compiler interessieren, schauen Sie sich einmal BASIC 64 von DATA BECKER an. Er übersetzt in reine Maschinensprache, liefert also sehr schnelle Ergebnisse und verfügt noch über weit mehr Möglichkeiten.

Ein letzter Tip zum Thema Compiler: Diese sind nicht nur interessant, wenn es um die Beschleunigung von Programmen geht, vielmehr bieten sie noch eine weitere interessante Möglichkeit:

Mit ihnen können leicht Programmierfehler gefunden werden. Stellen Sie sich vor, Sie haben in einem Programm ein fehlerbehaftetes Unterprogramm, das nur sehr selten aufgerufen wird. Bis zum ersten Aufruf bleibt der Fehler dann unentdeckt, und Sie geben das Programm mit diesem Fehler vielleicht weiter. Ein Compiler prüft das Programm komplett, und somit findet er auch den Fehler im Unterprogramm und meldet ihn - eine komfortable Möglichkeit zum Aufspüren von Fehlern.

So, wir hoffen, Ihnen hiermit noch einige gute Anregungen mit auf den Weg gegeben zu haben. Denn wir müssen uns jetzt natürlich langsam voneinander trennen und uns damit abfinden, daß Sie kein Einsteiger mehr sind. Aber da das ja unser gemeinsames Ziel war, überwiegt sicherlich dabei das lachende Auge.

Ein letztes noch: Sollten Sie bisher unsere Pannenhilfe im nächsten Kapitel nicht nötig gehabt haben, sollten Sie sie dennoch in Erinnerung behalten. Denn man kann ja nie wissen. Aber wenn wir schon wünschen, daß Sie auch in Zukunft dieses Buch noch gut gebrauchen können, denken wir natürlich lieber an das Lexikon am Schluß als an die Pannenhilfe.

11. Wenn etwas schiefgeht: Unser Pannen-Service

Natürlich können wir keinen telefonischen Beratungs- und handwerklichen Reparaturservice einrichten. Aber die Autoren eines Einsteiger-Buches können sich Gedanken machen über grundsätzliche Mißverständnisse und Unklarheiten, an denen sie als Schreiber des Buches zumindest indirekt die Schuld tragen. Denn wenn Sie uns nicht verstehen oder aufgrund einer unklaren Beschreibung etwas falsch machen, müssen wir das natürlich uns zuschreiben. Weil wir als Autoren aber auch nicht fehlerlos sind und Druckfehler sogar sowieso außerhalb unseres Einflusses liegen, haben wir in diesem Kapitel für Sie einige allgemeine Erste-Hilfe-Regeln zusammengestellt. Wir hoffen, daß dann nichts mehr schiefgeht.

Ganz allgemein gehen wir davon aus, daß Sie die Befehle und Programmzeilen, die Sie in diesem Buch finden, genau eingegeben haben. Um etwaigen Druckfehlern begegnen zu können, ziehen Sie im Zweifelsfall eine der vielen Kapitel-Zusammenfassungen und besonders auch das Lexikon ergänzend heran.

Kommen wir jetzt aber zu speziellen Fehlerquellen.

Panne: Programmänderungen werden nicht durchgeführt

Sie haben in einem Programm eine Zeile geändert, aber anschließend funktioniert das Programm nicht entsprechend der Änderung.

Lösung: Wahrscheinlich haben Sie das Drücken der Return-Taste vergessen. Lassen Sie sich mit LIST die geänderte Zeile zeigen, nehmen Sie die Änderung erneut vor, und betätigen Sie zum Abschluß die Return-Taste.

Panne: Cursor produziert Grafikzeichen

Sie wollen in einer Zeile eine Änderung vornehmen, aber der Cursor funktioniert nicht mehr wie normal, sondern produziert seltsame Grafikzeichen.

Lösung: Sie haben mit der INST-Taste Platz geschaffen oder ein Anführungszeichen verwendet. Dadurch befindet sich der C64 im sogenannten "Einfügemodus", bei dem die Cursor-Tasten eine andere Funktion haben. Drücken Sie Shift+Return, dadurch verläßt der Cursor die Zeile, ohne diese auszuführen, und Sie können erneut in die Zeile gehen und den Cursor dort normal bewegen.

Panne: PRINT ergibt statt eines Wortes eine 0

Sie wollen ein Wort auf den Bildschirm schreiben, aber der PRINT-Befehl gibt nicht das gewünschte Wort aus, sondern eine 0.

Lösung: Sie haben die Anführungszeichen vergessen. Wenn Sie etwas unverändert und ohne irgendeine Berechnung auf den Bildschirm ausgeben wollen, so muß es in Anführungszeichen stehen.

Panne: PRINT ergibt statt Variablenwert immer den Variablennamen

Der PRINT-Befehl gibt nicht den Wert der Variablen, sondern den Variablennamen aus. Sie haben beispielsweise in A$ den Namen "Peter" gespeichert, erhalten aber beim PRINT-Befehl immer "A$" auf dem Bildschirm.

Lösung: Versehentlich haben Sie ein Anführungszeichen gesetzt. Dadurch gibt BASIC das folgende ohne jede Änderung auf dem Bildschirm aus. Entfernen Sie das Anführungszeichen.

Panne: BASIC reagiert nicht auf Befehle im Direktmodus

Sie wollen im Direktmodus (also ohne Zeilennummer) einen Befehl ausführen lassen. BASIC führt diesen Befehl aber nicht aus, sondern setzt den Cursor nach Betätigung der Return-Taste einfach eine Zeile tiefer. Beispielsweise befehlen Sie

```
RUN
```

aber das Programm startet nicht.

Lösung: Wenn Sie richtig die Return-Taste betätigt haben und zu Beginn der Zeile auch wirklich keine Zahl steht (sonst Programm-Modus), dann gibt es nur eine Erklärung: Sie geben den Befehl RUN in einer verlängerten Programmzeile ein. BASIC kann nämlich auch mit Programmzeilen arbeiten, die zwei Bildschirmzeilen lang sind. Das passiert immer dann, wenn Sie bei der Eingabe einer Programmzeile über den rechten Bildschirmrand hinausschreiben. Selbst wenn Sie mit der Backspace-Taste die Zeichen in der zweiten Zeile löschen, gehört diese zweite Zeile immer noch als Programmzeile zur ersten. Machen Sie zweierlei:

1. Merken Sie sich die Nummer der Programmzeile, bewegen Sie den Cursor mit den Cursor-Tasten in die unterste Bildschirmzeile, und drücken Sie einige Mal die Return-Taste. Nun sind Sie sicherlich nicht mehr in einer erweiterten Programmzeile.
2. Lassen Sie sich mit LIST die gemerkte Zeile zeigen, und kontrollieren Sie, ob diese noch Reste Ihrer Befehlseingaben enthält - beispielsweise noch das RUN. Entfernen Sie diese Reste gegebenenfalls. Wenn Sie abschließend (nach der Änderung) Return drücken, springt der Cursor zwei Zeilen tiefer. Dort können Sie den gewünschten Befehl eingeben.

Noch ein Tip zu erweiterten Programmzeilen. Diese erkennt man eben daran, daß der Cursor beim Druck auf die Eingabetaste zwei Zeilen weiter nach unten springt.

Panne: Unerklärlicher SYNTAX ERROR im Direktmodus

Sie geben im Direktmodus (also ohne Zeilennummer) einen korrekten Befehl ein, aber BASIC meldet einen "Syntax error" oder eine andere Fehlermeldung, die Sie sich nicht erklären können.

Lösung: Wahrscheinlich befinden sich in der Zeile, in der Sie Ihre Eingabe machen, noch irgendwelche Zeichen, beispielsweise von einer vorherigen Bildschirmausgabe. Geben Sie einfach in der Zeile den gewünschten Befehl mit einem abschließenden Doppelpunkt ein, oder setzen Sie den Cursor in eine Zeile, in der keine Reste einer Bildschirmausgabe mehr vorhanden sind.

Panne: ?REDO FROM START beim INPUT-Befehl

Sie wollen in einem laufenden Programm hinter dem Fragezeichen eine Eingabe machen. Wenn Sie aber etwas eingeben, arbeitet das Programm nicht weiter, sondern es erscheint immer die Meldung:

```
?REDO FROM START
```

auf dem Bildschirm.

Lösung: Beim INPUT-Befehl muß ja stets eine Variable angegeben werden, die die Eingabe aufnehmen soll. In Ihrem Fall ist es eine numerische Variable, die nur Zahlen enthalten darf, Sie geben aber einen String (Zeichenkette) ein. Entweder haben Sie beim Variablennamen das $-Zeichen vergessen, oder Ihre Eingabe enthält Buchstaben und Sonderzeichen.

Panne: Scheinbar korrekte Programmzeile ergibt SYNTAX ERROR

Sie haben eine Programmzeile mit Variablen und starten das Programm. Der C64 meldet immer einen SYNTAX ERROR in der Zeile, obwohl diese völlig in Ordnung zu sein scheint.

Lösung: Sie haben versehentlich einen Befehl oder ein reserviertes Wort im Variablennamen verwendet, zum Beispiel ON in

TELEFON. Benutzen Sie einen anderen Variablennamen. Kurze Variablennamen sind zwar weniger ausdrucksstark, aber verhindern dieses Problem. Schauen Sie im Lexikon unter Variablennamen nach.

Panne: Offensichtlich unsinnige Werte bei Berechnungen

Sie wollen in einem Programm etwas berechnen lassen, doch der C64 kommt immer zu sehr eigenartigen Ergebnissen. Sie haben schon mehrfach mit dem Taschenrechner nachgerechnet und glauben eher diesem als dem Programm.

Beispiel:

```
10 KILO = 10
20 KILOMETER = 20
30 PRINT KILO/KILOMETER
```

Dies Programm ergibt als Ausgabe eine "1", obwohl eigentlich 0.5 herauskommen müßte.

Lösung: Die beiden Variablennamen unterscheiden sich für den C64 nicht, da er nur die ersten beiden Zeichen berücksichtigt. Für ihn steht dort:

```
10 KI = 10
20 KI = 20
30 PRINT KI/KI
```

Wählen Sie Variablennamen, die sich schon in den ersten beiden Zeichen unterscheiden, beispielsweise KI für KILO und KM für KILOMETER.

Panne: Programm versehentlich gelöscht

Sie haben ein kleines Programm geschrieben, nicht gespeichert.

Nun haben Sie versehentlich NEW eingegeben oder durch das Laden des Inhaltsverzeichnisses das Programm im Speicher gelöscht. Damit ist das Programm leider verloren.

Lösung: Trotzdem können Sie alle das, was noch auf dem Bildschirm sichtbar ist, leicht wiederholen, indem Sie einfach in jede noch sichtbare Programmzeile gehen und auf die Return-Taste drücken. Achtun: Im Falle des Inhaltsverzeichnisses **müssen** Sie vorher NEW eingeben, sonst bleiben Reste davon im Speicher.

Panne: Sie starten ein gerade neu geschriebenes Programm mit RUN und erhalten einen SYNTAX ERROR IN 0, obwohl Sie natürlich wie immer mit Zeile 10 begonnen haben.

Halt: .....

Lösung: Denken Sie einen Augenblick nach. Haben Sie vor dem Schreiben des Programms das Inhaltsverzeichnis der Diskette geladen und das anschließende NEW vergessen? Dann geben Sie jetzt kein LIST ein. Sie können den noch auf dem Bildschirm sichtbaren Teil des Programms retten, indem Sie NEW eingeben und anschließend alle Zeilen auf dem Bildschirm durch Return übernehmen.

Panne: TYPE MISMATCH ERROR in ...

Sie erhalten die Fehlermeldung: TYPE MISMATCH ERROR in 90.

Lösung: Dieser Fehler tritt auf, wenn Sie numerische Variablen mit Stringvariablen verwechseln oder gleichsetzen. Numerische Variablen können nur Werte von anderen numerischen Variablen zugewiesen bekommen, Zahlen oder das Ergebnis von Berechnungen aufnehmen. Stringvariablen können nur Zeichenketten (in Anführungszeichen) oder den Inhalt anderer Stringvariablen aufnehmen. So sind beispielsweise die folgenden Zuweisungen nicht erlaubt:

```
Eingabe = "Hallo" (Links numerisch, rechts String)
Eingabe$ = 4 (Links String, rechts numerisch)
Eingabe$ = Meineeingabe (Links String, rechts numerisch)
```

Panne: Programm will nicht stoppen

Sie haben ein Programm geschrieben, das nach dem Start gar nicht mehr aufhören will, weil es eine Endlosschleife enthält.

Lösung: Drücken Sie RUN/STOP, und BASIC stoppt das Programm. Nun sollten Sie natürlich vor einem neuen Start erst einmal das Programm so abändern, daß es auch normal beendet werden kann. Manchmal funktioniert RUN/STOP nicht, weil das Programm INPUT-Befehle enthält, und während der C64 das INPUT abarbeitet, reagiert er nicht auf RUN/STOP. Drücken Sie in einem solchen Fall gleichzeitig RUN/STOP + RESTORE.

Panne: NEXT WITHOUT FOR ERROR IN ...

Sie erhalten die Fehlermeldung NEXT WITHOUT FOR ERROR IN 30.

Lösung: Hier können mehrere Fehler vorliegen. Haben Sie überhaupt vor der Zeile 40 eine Zeile, die die Schleife beginnt? Wenn ja, so haben Sie sich vielleicht mit dem Variablennamen für die Schleife verschrieben. Zu einen "For I = 1 to 100" gehört auch ein "Next I" und kein "Next J". Eventuell verwenden Sie auch eine geschachtelte Konstruktion aus zwei Schleifen. Dann muß die zuletzt begonnene Schleife zuerst wieder beendet werden. Beispielsweise ist folgende verschachtelte Schleife nicht richtig:

```
10 For I = 1 to 10
20 For J = 1 to 10
30 Print I,J
40 Next I Rem (Hier muß erst die innere Schleife (J)
                     geschlossen werden )
50 Next J
```

Panne: RETURN WITHOUT GOSUB ERROR IN ...

Sie erhalten die Fehlermeldung RETURN WITHOUT GOSUB ERROR IN 30.

Lösung: Wenn BASIC auf ein "RETURN" trifft, so will es zu der Zeile zurückkehren, in der das zugehörige GOSUB gestanden hat. In diesem Fall ist BASIC aber auf ein "RETURN" gestoßen, ohne daß vorher GOSUB befohlen wurde. Haben Sie das GOSUB vergessen? Wenn nein, so gibt es noch eine häufige Fehlerquelle: Sie haben ein Hauptprogramm, das ein Unterprogramm aufruft. Das Hauptprogramm endet aber nicht mit einem END, und daher "läuft" BASIC irgendwann fälschlicherweise in das Unterprogramm. Dieser Fehler tritt beispielsweise in folgendem Programm auf:

```
10 REM Dies ist das Hauptprogramm
20 FOR I = 1 TO 100
30 GOSUB 100
40 NEXT I

100 REM Unterprogramm
110 PRINT I
120 RETURN
```

Fügen Sie die folgende Zeile ein:

```
50 END
```

Panne: UNDEF'D STATEMENT ERROR in ...

Sie erhalten die Fehlermeldung UNDEF'D STATEMENT ERROR in 30.

Lösung: Sie haben versucht, das Programm mit GOTO springen zu lassen, aber die angegebene Zeilennummer existiert gar nicht. Kontrollieren Sie auf Tippfehler, und schauen Sie nach, ob Sie die Zielzeile vielleicht gelöscht haben.

Panne: BAD SUBSCRIPT ERROR in ...

Sie erhalten nach dem Starten des Programms die Fehlermeldung: BAD SUBSCRIPT ERROR in 20.

Lösung: Sie arbeiten mit einem Feld und versuchen ein Element dieses Feldes anzusprechen, das außerhalb des durch DIM angegebenen Bereiches liegt. Beispiel:

```
10 DIM A(100)
20 A(200) = 4
```

Panne: Überraschendes OUT OF DATA ERROR

Völlig überraschend taucht nach Betätigung der Return-Taste obige Fehlermeldung auf, obwohl Sie in Ihrem Programm weder DATA-Zeilen noch den READ-Befehl verwenden.

Lösung: Immer, wenn der C64 mit etwas fertig ist, meldet er sich mit einem READY. Schauen Sie sich das READY noch einmal genau an. Der C64 interpretiert diese Zeile als READ Y. Sie haben versehentlich in dieser Zeile die Return-Taste betätigt und haben keine DATA-Zeilen, aus denen der C64 sein Y holen könnte.

Panne: FILE NOT FOUND ERROR

Sie wollen ein Programm oder das Inhaltsverzeichnis der Diskette laden und erhalten diese Fehlermeldung. Zusätzlich blinkt die rote Lampe der Floppy.

Lösung: Sie haben entweder keine Diskette eingelegt oder den Diskettenschacht nicht geschlossen, oder das Programm befindet sich nicht auf der Diskette (falsche Schreibweise). Beim nächsten richtigen Zugriff auf die Floppy hört das Blinken auf.

Panne: DEVICE NOT PRESENT ERROR

Sie wollen das Diskettenlaufwerk benutzen und erhalten diese Fehlermeldung.

Lösung: Sie haben vergessen, die Floppy einzuschalten. Sollte sich nach dem Einschalten nichts ändern, so schalten Sie alle Geräte aus, und überprüfen Sie die Verbindungskabel. Achtung: Nur im ausgeschalteten Zustand Verbindungskabel herausziehen oder einstecken.

Panne: Floppylampe blinkt beim Speichern

Sie speichern ein Programm, und der C64 meldet auch die Ausführung mit READY, aber die Lampe der Floppy blinkt.

Lösung: Sie haben versucht, ein bereits unter demselben Namen bestehendes Programm noch einmal abzuspeichern. Löschen Sie entweder das alte Programm, oder verwenden Sie den Klammeraffen oder aber wählen Sie einen anderen Namen. Nähere Informationen zum Klammeraffen finden Sie im Lexikon unter SAVE "PROGRAMMNAME",8 (Return).

Panne: Speichern ohne Erfolg

Sie haben ein Programm in einer neuen Version gespeichert. Aber beim späteren Laden entpuppt es sich als eine ältere Version.

Lösung: Sie haben es versehentlich unter demselben Namen zu speichern versucht und nicht gemerkt, daß die Floppylampe blinkte, weil Sie anschließend sofort das Inhaltsverzeichnis geladen haben. Schauen Sie unter SAVE "PROGRAMMNAME",8 (Return) im Lexikon nach.

Wir hoffen, daß wir die wesentlichen Fehlerquellen aufgezeigt haben. Sollten noch weitere Probleme auftauchen, so können Sie Fehlermeldungen und deren Erklärung in Ihrem C64-Handbuch nachschlagen.

Lexikon zum BASIC des C64

Das folgende Lexikon ist zum schnellen Nachschlagen gedacht. Auf diese Weise wird mühsames Suchen nach Begriffen und Problemlösungen im Text vermieden. Außerdem bleibt dieses Buch so für Sie auch noch eine Hilfe, wenn Sie schon lange kein Einsteiger mehr sind. Wir haben uns in der Regel für eine möglichst knappe Darstellung entschieden, um das Ganze übersichtlich zu halten. Sollten zu speziellen Fragen noch weitere Informationen nötig sein, die über den Rahmen dieses Einsteiger-Buches hinausgehen, dann können Sie auf die weiterführende Literatur zurückgreifen, die wir Ihnen vorgestellt haben.

Abbruch eines Programms

RUN/STOP-Taste oder, wenn das nicht hilft, dann diese Taste gedrückt halten und zusätzlich die RESTORE-Taste drücken.

Alphanumerische Variablen

Unter alphanumerischen Variablen versteht man solche veränderliche Platzhalter oder Schubladen in Programmen, die aus beliebigen Zeichen, also besonders auch Buchstaben (daher auch "alpha...") bestehen können. Sie können auch Zahlen enthalten (daher "...numerisch"), die dann aber auch als Zeichen aufgefaßt werden. Alphanumerische Variablen werden ->Stringvariablen genannt, denn unter ->Strings versteht man beliebige Zeichenketten, die keine Rechenwerte darstellen. Stringvariablen müssen am Schluß ein Dollarzeichen enthalten. Der ihnen zugewiesene Ausdruck steht in Anführungszeichen:

```
NAME$ = "Benzinverbrauchs-Programm"
```

Zu beachten ist, daß der C64 bei Variablen nur die ersten beiden Zeichen unterscheidet, d.h. NAME$, NAME1$ und NA$ sind für ihn identisch.

Array

Siehe ->Feld.

ASCII-Grafik

Von ASCII-Grafik wollen wir im Rahmen dieses Buches sprechen, wenn man versucht, mit einfachsten Mitteln Bilder auf den Bildschirm zu bekommen. Zu den einfachsten Mitteln gehören die sogenannten ASCII-Zeichen, die bei jedem C64 sofort zur Verfügung stehen, und außerdem Grafikzeichen, die durch gleichzeitiges Drücken der Shift-Taste und der Commodore-Taste erreicht werden. So kann man mit der Taste über dem Gleichheitszeichen einmal ein Sternchen, zum anderen einen mittelbreiten waagerechten Strich und zum Schluß noch ein Dreieck erzeugen und als Bestandteil einer Grafik verwenden.

ASCII-Zeichen

ASCII ist die Abkürzung für "American Standard Code for Information Interchange", d.h. für genaue Absprachen darüber, welche Zahl der C64 einem Zeichen zuordnet. Denn unser C64 kann natürlich nur in Zahlen "denken" und muß jedes über die Tastatur oder sonstwie an ihn übergebene Zeichen erst in eine solche Zahl übersetzen. Die Ziffern beginnen zum Beispiel bei der Nummer 49 (CHR$(49)=1), die Buchstaben bei der Nummer 65 (CHR$(65)="A").

?BAD SUBSCRIPT ERROR IN ...

Diese Fehlermeldung taucht im Zusammenhang mit Feldern (siehe ->DIM und ->Feld) auf und besagt, daß die Elementangabe außerhalb des definierten Bereichs liegt. Hat man MO$ mit 12 dimensioniert, führt die Frage nach MO$(13) zu dieser Fehlermeldung.

BASIC

BASIC ist eine Programmiersprache. Im Gegensatz zu Anwenderprogrammen, die meist nur eine Lösung für einen Problembereich darstellen und nicht geändert werden können (Textverarbeitung), dienen Programmiersprachen dazu, beliebige Probleme lösbar zu machen. Dazu stehen eine Vielzahl von Befehlen zur Verfügung, aus denen jeder das Programm für sein Problem erstellen kann. Das Besondere beim C64 ist, daß BASIC sofort nach dem Einschalten da ist und das eigentliche Instrument des

Umgangs mit dem Computer darstellt, um das man gar nicht herumkommt.

Bildschirmfarben (einstellen)

Mit einem kleinen Programm können Sie die Bildschirmfarbe ausprobieren, die Ihnen am besten gefällt. Dabei wird auf den POKE-Befehl und eine ganz bestimmte Speicherstelle zurückgegriffen. Dort werden verschiedene Zahlen ausprobiert (I) und jeweils auf dem Bildschirm angezeigt. Mit Hilfe der RUN/STOP-Taste kann man dann das Programm unterbrechen und sieht die Nummer der gewünschten Bildschirmfarbe. Diese kann man dann im Direktmodus eingeben oder in ein Programm aufnehmen.

```
10 PRINT "CLR"
20 FOR I = 1 TO 16
30 PRINT I
50 POKE 53281,I
60 FOR J = 1 TO 1000 : NEXT J: REM VERZÖGERUNGSSCHLEIFE
70 NEXT I
```

Bildschirm löschen

Im Direktmodus durch die Kombination aus Shift und CLR/Home-Taste. Im Programm-Modus durch den PRINT-Befehl, verbunden mit einem String, der das Zeichen enthält, das man beim Betätigen der Shift-CLR-Kombination erreicht, eine Art dunkles Herzchen auf hellem Grund.

```
10 PRINT "CLR"
```

CLR steht hier für Shift plus CLR/Home-Taste.

Binär

Mit binär meint man eine Darstellungsweise, die sich nur aus zwei Ziffern (0,1), den sogenannten ->Bits, zusammensetzt. Im Gegensatz zum dezimalen Zahlensystem mit seinen 10 Ziffern (0 - 9) ist das binäre Zahlensystem dem internen Aufbau eines Computers angepaßt. So wie sich größere Zahlen im Dezimalsystem zusammensetzen lassen, werden auch im Binärsystem Zahlen aus allerdings mehr Ziffern zusammengesetzt.

Bit

Ein Bit stellt die kleinste Einheit im ->binären Zahlensystem dar. Es kann nur zwei Zustände annehmen (0,1) und damit nur zwei verschiedene Informationen speichern. Um beispielsweise Zeichen zu speichern, müssen mehrere Bits zu einem ->Byte zusammengefaßt werden.

Byte

Eine Gruppe von 8 Bits nennt man Byte. Während ein ->Bit nur die zwei Zustände 0 und 1 annehmen kann, können in einem Byte 0 - 255 Zustände gespeichert werden. Meist wird ein Zeichen in einem Byte abgelegt, so daß Sie die vielen Angaben in Bytes auch in "Anzahl Zeichen" übersetzen können.

CHR$(X)

Bei dieser Funktion handelt es sich um das Gegenstück zur ASC(X$)-Funktion. Bei CHR$(X) steht das X für den ASCII-Wert, und man bekommt das zugehörige Zeichen. Der Befehl

```
PRINT CHR$(65) (Return)
```

ergibt zum Beispiel den großen Buchstaben A. Eine entsprechende Tabelle finden Sie im Handbuch des C64, oder Sie lassen sich einige Werte am Bildschirm anzeigen.

CLR

Dieser Befehl löscht nach dem Durchlauf eines Programms nur die Werte für die Variablen, während das Programm erhalten bleibt.

Computersprachen

Siehe ->BASIC.

CONT

Siehe ->STOP.

CONTROL-Taste

Siehe ->CTRL-Taste.

CTRL-Taste

Mit dieser Taste links neben dem Buchstaben Q kann die Anzeige eines Programmlistings (der Programmzeilen) verlangsamt werden. Dasselbe gilt für die Anzeige des Inhaltsverzeichnisses einer Diskette.

Cursor

Engl. Läufer: Kleiner quadratischer blinkender Lichtfleck auf dem Bildschirm, der die Stelle markiert, an der Sie mit Hilfe der Tastatur ein Zeichen plazieren können. Mit den sogenannten Cursor-Tasten (das sind die beiden Tasten unter der Return-Taste, z.T. in Verbindung mit einer der Shift-Tasten) können Sie diesen Cursor auf dem Bildschirm frei hin- und herbewegen.

Cursor-Bewegung

Siehe ->Cursor.

Cursor-Tasten

Das sind die beiden Tasten, mit denen der ->Cursor in die vier möglichen Richtungen bewegt werden kann. Für zwei Richtungen muß man gleichzeitig die Shift-Taste gedrückt halten.

DATA

Siehe ->READ

Datei

Zusammengehörende Datenmenge, z.B. Programm, Text, Grafik u.a., erscheint im Inhaltsverzeichnis einer Diskette als eigener Eintrag mit der Größe (in Blöcken), dem Namen (in Anführungszeichen, maximal 16 Zeichen) und einer Kurzinformation über die Art der Datei (bei BASIC-Programmen immer PRG).

Dateinamen

Unter Dateinamen versteht man die Buchstaben- und Zahlengruppen, unter denen eine Datei im Inhaltsverzeichnis abgespeichert wird. Er kann maximal 16 Zeichen lang sein. Ist er länger, werden die überschüssigen Zeichen einfach nicht be-

rücksichtigt. Dateinamen können auch Leerzeichen oder Grafikzeichen enthalten.

Del-Taste

Mit dieser Taste (zweite Taste oberhalb der Return-Taste) löschen Sie das Zeichen, auf dem gerade der Cursor steht, die rechts davon stehenden Zeichen werden entsprechend nach links verschoben.

DIM

Mit dem DIM-Befehl können Sie die Größe eines ->Feldes, d.h. die Anzahl der Elemente festlegen, die zu einer Teilvariablen gehören. DIM MO$(12) reserviert Ihnen z.B. 12 variable Monatsnamen, von Januar, das wäre MO$(1), bis zum Dezember, das wäre MO$(12). Ein zweidimensionales Feld könnte zum Beispiel mit DIM MO$(12,100) dimensioniert werden. Das würde dann ermöglichen, weiterhin die 12 Monatsnamen als Elemente zu verwenden. Als zweite Unterscheidungsmöglichkeit hätten wir aber etwa 100 Jahresnamen, von 1901 bis 2000. (Wir wollen die Jahreszahlen hier als Namen verstehen.) MO$(1,1988) stünde dann für den Januar des Jahres 1988. Die höchstmögliche Dimensionierung liegt bei 255, aber das kann man sich kaum noch vorstellen, und zeichnerisch, etwa in einer Tabelle, ist es schon gar nicht mehr darstellbar. Die Zahl der Elemente pro Teilvariable ist auf 32767 begrenzt. Aber auch das ist für uns keine Grenze, über die wir klagen müßten.

Mit einem DIM-Befehl können mehrere Felder definiert werden:

```
DIM MO$ (12,100), WO$ (7), MT (31)
```

Dabei stünde WO$ für Wochentag und MT für Monatstag.

Wichtig ist,

- daß der DIM-Befehl vor der ersten Benutzung des Feldes gegeben wird,
- daß ein Feld nur einmal dimensioniert werden kann. Sonst gibt es die Fehlermeldung: "?REDIM'D ARRAY ERROR in ...",

- daß man nicht versucht, ein Element außerhalb der Dimensionierung zu verwenden: Bei DIM MO$ (13) führt der Befehl PRINT MO$ (13) zu der Fehlermeldung: "BAD SUBSCRIPT ERROR IN ...", was soviel heißt, daß die Elementangabe außerhalb des definierten Bereichs liegt.

Division

Ausdruck für Teilung, Division.

?DIVISION BY ZERO ERROR IN ...

Diese Fehlermeldung zeigt an, daß Sie versucht haben, BASIC dazu zu bringen, durch Null zu dividieren. Das geht in der Mathematik nicht, und es geht natürlich auch nicht in BASIC. Häufig wird man deshalb mit dieser Meldung überrascht, weil die problematische Null nicht direkt, sondern erst als Ergebnis einer Berechnung o.ä. erscheint. Schließen Sie die Null einfach durch eine Bereichsfestlegung aus (IF ZAHL = 0 THEN ...).

Drucken von Programmzeilen

Umlenken der Daten vom Bildschirm auf den Drucker.

Hierzu muß zunächst ein Befehlskanal zum Drucker geöffnet werden (OPEN1,4), sodann muß der Befehl kommen, die Daten vom Bildschirm auf den Drucker umzulenken (CMD1). Dann kommt der ganz normale LIST-Befehl, den man auch spezifizieren kann: LIST 100-300. Am Schluß müssen beide Anfangsbefehle in umgekehrter Reihenfolge wieder aufgehoben werden. Das geschieht durch PRINT#1 und CLOSE1. Die komplette Folge der Direktbefehle lautet (mit Return abgeschlossen):

```
OPEN1,1
CMD1
LIST
PRINT#1
CLOSE1
```

Drucken von Programmergebnissen

Auch dies kann mit Hilfe der Befehle geschehen, die wir schon beim ->Drucken von Programmzeilen vorgestellt haben. Nur, daß jetzt die entsprechenden Befehle ins Programm eingebaut

und um den PRINT-Befehl herum angeordnet werden. Also zum Beispiel:

```
100 OPEN1,4
110 CMD1
120 PRINT "DIE ZAHL IST:"
130 PRINT A
140 PRINT#1
150 CLOSE1
```

Direktmodus

BASIC bietet die Möglichkeit, Befehle direkt auf den Bildschirm zu schreiben. Nach Betätigen der Return-Taste werden sie ausgeführt. Anschließend ist der Vorgang abgeschlossen und nur dadurch wiederholbar, daß man mit dem Cursor noch einmal in die entsprechende Zeile geht und wiederum die Return-Taste betätigt.

Sinnvoll wäre etwa eine kleine Rechenaufgabe (Mehrwertsteuer z.B.):

```
PRINT 125.6/100*14 (Return)
```

Anschließend erscheint 17.584.

Wichtig kann der DIREKTMODUS auch sein, um sich am Schluß eines Programms zur Überprüfung Zwischenwerte, die nicht auf dem Bildschirm erscheinen, ausgeben zu lassen. Auch nach dem Ende eines Programms bleiben nämlich die Werte der Variablen solange erhalten, bis ein NEW oder CLR eingegeben oder eine Zeile des Programms geändert wird.

Das Gegenstück zum ->Direktmodus ist der ->Programm-Modus.

Einfügen von Text in eine Befehls- oder Programmzeile

Siehe ->INST-Taste.

END

Mit diesem Befehl können Sie in einem Programm jederzeit die weitere Abarbeitung beenden. Wichtig ist dieser Befehl vor allem zu Beginn von Unterprogrammen, sonst werden sie unbeabsichtigt, unabhängig von einem GOSUB-Befehl, abgearbeitet.

Exponentialschreibweise

Siehe ->Potenzieren.

Feld

Unter einem Feld oder ->Array versteht man Variablen, die sich aus mehreren variablen Bestandteilen, also Einzelvariablen zusammensetzen. Der Vorteil einer solchen Gruppe von Variablen besteht darin, daß man auf sie mit einer Schleife zugreifen kann. Nehmen wir eine Variable MO$, und dimensionieren wir sie mit 12. Dies geschieht durch die DIM-Anweisung: DIM MO$ (12).

```
10 DIM MO$(12)
20 FOR I = 1 TO 12
30 READ MO$(I)
40 NEXT I
50 DATA Januar, Februar, Maerz, April, Mai, Juni, Juli, August,
September, Oktober, November, Dezember
```

Am Ende versteht das Programm unter MO$(3) immer den Maerz und unter MO$(10) immer den Oktober.

Solche Felder können auch mehrdimensional sein. Ein zweidimensionales Feld könnte man sich in diesem Fall so vorstellen, daß in senkrechten Zeilen die Monate und in waagerechten Spalten verschiedene Jahre eingetragen werden. Für die Jahre 1987, 1988, 1989 sähe das so aus:

```
10 DIM MO$(12,3)
```

Hier wird festgelegt, daß die Variable über 12 Monatsnamen und drei Jahresnamen verfügen soll. Anschließend kann man die Daten so aufbauen, daß MO$(11,2) der November des zweiten Jahres, also in diesem Falle: der November 1988 ist usw.

Felder können über noch mehr Dimensionen verfügen, die Vorstellung oder auch grafische Darstellung ist dann aber natürlich nicht mehr so einfach.

Siehe auch ->DIM.

File

Anderer Begriff für ->Datei.

FOR NEXT

Der Doppelbefehl FOR NEXT führt zu einer sogenannten Programmschleife. In der Startzeile der Schleife steht zum Beispiel FOR I = 1 TO 10. In der Zielzeile der Schleife steht dann NEXT I. Das, was zwischen diesen beiden Zeilen steht, wird dann zehnmal abgearbeitet, wobei jedesmal I um eins erhöht wird. Das heißt: Der Computer zählt seine Schleifendurchgänge mit, und sobald er die gesetzte Grenze erreicht hat, verläßt er den Schleifenbereich und macht dahinter weiter.

Fragezeichen als Meldung des C64

Das Fragezeichen taucht immer dann auf, wenn BASIC einen INPUT-Befehl abarbeitet. Da ein Fragezeichen aber nicht sehr aussagekräftig ist, sollten Sie vorher einen Hinweis auf den Bildschirm geben, um was für eine Eingabe es sich jetzt handelt, also entweder

```
10 PRINT "Bitte Monat eingeben"
20 INPUT MO$
```

oder noch kürzer und eleganter

```
10 INPUT "Bitte einen Monat eingeben", MO$
```

Formatieren einer neuen Diskette

Vor der ersten Benutzung einer Diskette muß sie speziell für die Benutzung durch den C64 und seine Floppy vorbereitet werden. Dies geschieht durch den etwas kompliziert aussehenden Direktbefehl (genau genommen sind es zwei, was der Doppelpunkt dazwischen zeigt):

```
OPEN 1,8,15,"N:DISKETTENNAME,01":CLOSE1 (plus Return-Taste)
```

Dabei kann der Diskettenname frei gewählt werden (maximal 16 Zeichen). Die darauffolgende ->ID-Nummer (hier: 01) kann bis 99 gehen.

Wendet man den Formatierbefehl auf eine bereits benutzte Diskette an, so gehen alle Daten auf ihr unwiederbringlich verloren. Von daher sollte man mit diesem Befehl sehr vorsichtig umgehen. Sicherheitshalber bei bereits benutzten Disketten immer erst ins Inhaltsverzeichnis sehen.

GET

Mit diesem Befehl werden einzelne Zeichen von der Tastatur geholt. Das entsprechend eingegebene Zeichen wird einer Variablen zugeordnet. Interessant ist dieser Befehl vor allem für eine sogenannte Tastaturabfrage, bei der das Programm erst weiterläuft, wenn eine Taste gedrückt wird:

```
100 GET A$
110 IF A$ = "" THEN 100
```

In diesem Falle bleibt das Programm solange in der Endlosschleife 100-110, bis eine beliebige Taste gedrückt wird. Dann ist nämlich A$ nicht mehr "", und die nächsten Zeilen werden abgearbeitet.

GOSUB

Der Befehl GOSUB 500 löst zunächst ebenso wie GOTO 500 einen Sprung in die Programmzeile 500 aus. Der Unterschied und Vorteil besteht aber darin, daß sich BASIC die Absprungstelle merkt und am Ende des GOSUB-Unterprogramms zu ihr zurückkehrt. Am Ende findet BASIC nämlich den Befehl RETURN, der es zur Rückkehr auffordert. Nach einem GOTO-Sprung würde das Programm einfach an der Zielstelle weiterlaufen. Der besondere Vorteil der Kombination von GOSUB und RETURN besteht gerade darin, daß man häufig benutzte Routinen (etwa eine Umrechnung) in ein GOSUB-Unterprogramm verlegt, wo sie von verschiedenen Stellen des Programms schnell mal eben angesprungen werden können.

GOTO

Der Befehl GOTO 20 bedeutet, daß BASIC, sobald es zu diesem Befehl gelangt, nicht Zeile für Zeile weiterläuft, sondern zur Programmzeile 20 springt und dort weitermacht. Diese Programmzeile muß natürlich auch vorhanden sein, sonst gibt es eine Fehlermeldung.

Grundrechenarten

Durch die Benutzung der Zeichen für die Grundrechenarten kann addiert (+), subtrahiert (-), multipliziert (*) und dividiert (/) werden. Im letzteren Falle ist zu beachten, daß BASIC als Zeichen für die Division nicht den sonst gebräuchlichen Doppelpunkt, sondern den Schrägstrich nimmt. Der Doppelpunkt hat eine andere Funktion, nämlich mehrere Befehle in einer Zeile zu trennen.

Hauptprogramm

Besteht ein Programm aus mehr als nur einigen Programmzeilen, so ist es sinnvoll, dieses Programm in ein Hauptprogramm und mehrere ->Unterprogramme zu unterteilen. Dadurch wird es übersichtlicher (strukturiertes Programmieren). Am besten schreiben Sie in die Programmzeilen des Hauptprogramms, in denen Unterprogramme aufgerufen werden, mit ->REM einen Hinweis auf die Tätigkeit des Unterprogramms. Ein Beispiel für diese Programmstruktur bietet das Programm "Balkengrafik".

Hochpfeil

Hiermit ist eine Taste links oberhalb der Return-Taste gemeint, die einen nach oben gerichteten Pfeil produziert und für die Darstellung von Potenzen benötigt wird. Da dieses Zeichen schwer darzustellen ist, verwenden wir den Begriff Hochpfeil. Wo dieser Begriff in einer Befehlszeile auftaucht, ist das Zeichen gemeint, das die Taste produziert.

IF THEN

Der Befehl IF THEN besteht eigentlich aus zwei Befehlswörtern. Der erste Bestandteil (IF) prüft eine Bedingung, der zweite (THEN) nennt einen Befehl, der "bei bestandener Prüfung" aus-

geführt werden soll. Wichtig sind zum Beispiel Abfragen wie INPUT EINGABE$ und die anschließende Bedingungszeile:

```
IF EINGABE$ = "J" THEN GOTO 20.
```

Wichtig in diesem Zusammenhang sind die folgenden "Vergleichsoperatoren", die dem Vergleich von Daten dienen:

- das Gleichzeitszeichen: IF A = 20 THEN GOTO 100

- das Größer-als-Zeichen: IF A > 20 THEN GOTO 100

- das Kleiner-als-Zeichen: IF A < 20 THEN GOTO 100

Möglich sind auch Kombinationen:

- Kleiner oder gleich: IF A <= 20 THEN GOTO 100

- Größer oder gleich: IF A >= 20 THEN GOTO 100

- ungleich: IF A <> 20 THEN GOTO 100

Inhaltsverzeichnis der Diskette laden

Das Inhaltsverzeichnis einer Diskette wird mit folgendem Befehl in den Programmspeicher des C64 geladen:

```
LOAD"$",8 (Return)
```

Anschließend bekommt man es auf dem Bildschirm angezeigt mit:

```
LIST (Return)
```

Bei längeren Inhaltsverzeichnissen kann man man den Zeilendurchlauf mit Hilfe der CTRL-Taste verlangsamen. Abbrechen kann man ihn durch Drücken der RUN/STOP-Taste.

Zu beachten ist, daß beim Laden des Inhaltsverzeichnisses einer Diskette ein eventuell vorher geladenes Programm gelöscht wird. Man kommt in diesem Fall nicht darum herum, erst das Pro-

gramm abzuspeichern, dann das Inhaltsverzeichnis zu laden und schließlich bei Bedarf noch einmal das Programm zu laden.

Initialisierungsteil

Sinnvoll ist es, bei Programmen die Variablen und mögliche Anfangswerte in den ersten Programmzeilen festzulegen. Diesen Teil des Programms nennt man dann Initialisierungsteil. Dadurch werden Änderungen am Programm erheblich vereinfacht.

INPUT

INPUT VERBRAUCH: Der hinter dem erscheinenden Fragezeichen eingegebene Wert wird in der Variablen VERBRAUCH gespeichert. Es ist möglich, vor dem Fragezeichen einen erläuternden Text auszugeben, indem man diesen in Anführungszeichen setzt und durch ein Semikolon abtrennt, also etwa:

```
INPUT "Bitte den Verbrauch eingeben"; VB
```

Wird bei dem INPUT-Befehl eine numerische Variable verlangt (also eine Zahlenvariable), so muß auch die Eingabe numerisch sein, andernfalls erscheint die Fehlermeldung

```
? Redo from start
```

und die Eingabe muß wiederholt werden.

INPUT#

Dieser Befehl funktioniert grundsätzlich genauso wie INPUT, holt sich die Daten aber nicht über die Tastatur, sondern von Diskette. Hinter dem "#" muß noch eine Zahl - bei uns die 5 - stehen. Dabei muß die entsprechende Datei auf der Diskette vorher mit OPEN und ebenfalls der 5 geöffnet worden sein (siehe auch ->sequentielle Dateien).

Inst-Taste

Mit der INST-Taste können Sie das Zeichen, auf dem der Cursor steht und alles rechts davon, nach rechts verschieben und sich somit Platz für Einfügungen schaffen. Wichtig ist, daß Sie in den frei gewordenen Platz zwar dann beliebige Zeichen einfügen, nicht aber die Cursor-Tasten verwenden können. In die-

sem Fall erscheinen Grafikzeichen. In solch einem Fall kann man mit Shift plus Return-Taste problemlos aus der problematischen Zeile herauskommen und neu hineingehen.

Initialisierung

Der Teil eines Programms, der gewissermaßen den Beginn oder den Einstieg darstellt. In ihm werden z.B. Variablen definiert.

INT (X)

Mit Hilfe der INT-Funktion bekommen Sie Werte immer als ganze Zahlen: INT (5.23) ist eben die glatte Zahl 5. BASIC läßt also den Nachkommateil weg, d.h. rundet ab.

Integervariablen

Eine Untergruppe der ->Numerischen oder Zahlenvariablen stellen die Integervariablen dar. Sie verstehen alle ihnen zugewiesenen Zahlenwerte als ganze Zahlen. Dabei können sie von -32768 bis +32768 reichen. Kennzeichnend für Integervariablen ist das Prozentzeichen am Schluß. Die Zuweisung

```
KOSTEN% = 34.75
```

bedeutet also, daß das Programm diese Variable immer mit dem Wert 34 füllt und die Nachkommastellen entfallen läßt. Siehe auch INT(X).

Kopieren von Programmzeilen

Siehe ->Programmzeilen kopieren.

LIST

Zeigt die bisher eingegebenen Programmzeilen nach Zeilennummern geordnet auf dem Bildschirm an. Dabei kann die Zeile mit neuem Inhalt überschrieben werden, und die Zeile wird in der neuen Form nach dem Drücken der Return-Taste endgültig geändert.

Wird nach LIST eine Zeilennummer angegeben, so wird nur diese Zeile angezeigt. LIST 100 - 200 zeigt entsprechend alle Programmzeilen in den angegebenen Grenzen. Durch Drücken

der CTRL-Taste kann man den Durchlauf der Programmzeilen, das sogenannte ->Scrollen, verlangsamen.

Laden eines Programms

Siehe ->LOAD.

LOAD

Ein BASIC-Programm wird von Diskette geladen mit:

```
LOAD"PROGRAMMNAME",8 (Return)
```

Es steht dann im Programmspeicher zur Verfügung, kann mit LIST angesehen und verändert bzw. mit RUN in Gang gesetzt werden.

Auf eine komfortablere Möglichkeit, ein Programm zu laden, sei hier noch hingewiesen. Besonders bei längeren Programmnamen ist das Tippen eine lästige Angelegenheit. Man kann es sich ersparen, wenn man vorher das Inhaltsverzeichnis so lädt, daß der Dateiname auf dem Bildschirm zu sehen ist (ggf. mit RUN-STOP-Taste den Durchlauf der Zeilen stoppen). Anschließend löscht man den Programmspeicher wieder mit NEW (Return). Nun kann man bequem mit dem Cursor in die entsprechende Zeile des Inhaltsverzeichnisses gehen und den dort bereits stehenden Namen nutzen. Dabei muß man natürlich ein LOAD davorsetzen und hinter die zweiten Anführungszeichen ein ,8 setzen. An den Schluß der Ladezeile kommt ein Doppelpunkt, damit das hinterstehende PRG keine Schwierigkeiten macht. Nach Betätigung der Return-Taste wird dann das angewählte Programm ganz normal geladen. Vor dem anschließenden RUN oder LIST sollte man allerdings den Cursor in eine freie Bildschirmzeile bewegen oder den Bildschirm insgesamt löschen.

Löschen des Bildschirms

Im ->Direktmodus: CLR-Taste (mit Shift)

Im ->Programm-Modus: 10 PRINT "CLR"

CLR steht für das Zeichen, das sich ergibt, wenn man die CLR-Taste drückt, eine Art Herzchen auf weißem Grunde.

Löschen eines Programms auf Diskette

Der Befehl für das Löschen eines Programms auf Diskette lautet:

```
OPEN1,8,15,"S:PROGRAMMNAME":CLOSE1
```

Löschen des Programmspeichers

Siehe ->NEW.

Löschen von Programmzeilen

Siehe ->Programmzeilen löschen.

NEW

Vor der Erstellung oder dem Laden eines neuen Programms sollte man sicherheitshalber den Programmspeicher mit NEW (Return) frei machen, es kann sonst zu Überlagerungen kommen. Dies gilt besonders auch für den Fall, daß man das Inhaltsverzeichnis geladen hat. Dieses liegt nämlich wie ein BASIC-Programm im Programmspeicher und muß dort erst gelöscht werden, bevor man an einem anderen Programm arbeiten kann.

?NEXT WITHOUT FOR ERROR

Taucht diese Fehlermeldung auf, so bedeutet das, daß das Programm den Teilbefehl NEXT entdeckt hat, ohne vorher den ersten Bestandteil FOR gefunden zu haben. Dies taucht am ehesten in verschachtelten Schleifen auf, wo man leicht die Übersicht verliert, oder Sie haben die Zeile mit dem ersten Befehlsbestandteil, in der die Schleife geöffnet bzw. begonnen wird, gelöscht.

Numerische Variablen

Siehe ->Variablen.

PEEK (X)

Gegenstück zum POKE-Befehl. Hier erfährt man den Inhalt einer Speicherstelle.

POKE X,Y

Mit dem Befehl POKE X,Y können Sie direkt Werte in eine der ca. 64000 Speicherstellen des C64 schreiben. Dabei steht X für die Speicherstelle und Y für den Inhalt der Speicherstelle. Dieser kann von 0 bis 255 reichen. Mit POKE 53265,187 zum Beispiel schalten Sie um auf hochauflösende Grafik, und ab dann entzieht sich der C64 weitgehend Ihrer Kontrolle. Aus dem Grunde sollten Sie am Anfang sehr zurückhaltend mit POKE-Befehlen sein und diese nur dort verwenden, wo sie etwa in diesem Buch ihren festen Platz haben.

Verwendet werden können POKES etwa auch zum Einstellen der Bildschirmfarbe, siehe dort.

Potenzieren

Das Potenzzeichen in BASIC ist ein nach oben gerichteter Pfeil (Taste über der Return-Taste), den wir hier nicht abdrucken und darum als ->Hochpfeil in Formeln einbringen werden (2HOCHPFEIL2 ergibt beispielsweise 4). Ein Beispiel für den konkreten Einsatz: Die Programmzeile

```
10 PRINT 2HOCHPFEIL4
```

ergibt die Zahl 16.

PRINT (ersetzbar durch das Fragezeichen: ?)

Im Gegensatz zur englischen Wortbedeutung "drucke" bezieht sich dieser Befehl auf Ausgaben auf den Bildschirm, was am Anfang etwas mißverständlich und irreführend ist.

Bezogen werden kann der Befehl PRINT zum einen auf die Werte von Variablen (PRINT EINGABE bzw. ?E) oder auf Berechnungen. PRINT 5*7 ergibt dann auf dem Bildschirm das Ergebnis 35. Außerdem können Sie mit PRINT "NAME" sich jede beliebige Buchstabenkombination oder auch Strings (hier NAME) auf dem Bildschirm ausgeben lassen. Dasselbe gilt natürlich auch für vorher definierte Stringvariablen:

```
10 Name$ = "Peter"
20 PRINT Name$
```

Hier dürfte deutlich geworden sein, daß es sich bei PRINT um den wohl am häufigsten benutzten BASIC-Befehl handelt, wobei natürlich vor allem der ->Programm-Modus interessant ist. Aber auch im ->Direktmodus ist der Befehl hilfreich, um sich etwa nach Durchlauf eines Programms Zwischenwerte anzeigen zu lassen, die nicht auf dem Bildschirm auftauchten, aber noch im Speicher des Rechners enthalten sind.

Weil der PRINT-Befehl so eine grundsätzliche Bedeutung für jedes BASIC hat, kann man ihn beim Schreiben durch das Fragezeichen ersetzen. D.h. der Befehl ?SUMME ist identisch mit PRINT SUMME. Im Programm-Modus werden die Fragezeichen automatisch in PRINT-Befehle umgewandelt.

Ist die im PRINT-Befehl abgefragte Variable noch nicht mit einem Wert versehen, wird bei numerischen eine Null, bei Stringvariablen nichts ausgegeben.

Wichtig sind noch einige Hinweise zur Formatierung von Ausgaben: Will man mehrere Angaben in einer Zeile ausgegeben haben, muß man sie durch je ein Semikolon trennen:

```
PRINT FAKTOR1;FAKTOR2;FAKTOR3 (Return)
```

führt (bei entsprechenden Werten) dann zu der Ausgabe:

```
14 17 4 .
```

Eine andere Möglichkeit stellt das Komma zwischen Variablen in einem PRINT-Befehl dar. Durch das Komma beginnt die nächste Ausgabe auf den Bildschirm in ganz bestimmten Spalten, ähnlich der Wirkung der Tabulatortaste.

PRINT#

Grundsätzlich entspricht dieser Befehl dem PRINT-Befehl. Er bezieht sich auf ein anderes Gerät, etwa den Drucker, oder auf eine Datei auf Diskette. In beiden Fällen muß vorher der OPEN-Befehl angewendet worden sein. Nach PRINT# steht immer zunächst eine Zahl - bei uns stets die 5 - , die dem OPEN-Befehl entspricht, dann ein Komma und anschließend das, was ausgegeben werden soll. Wichtig ist, daß dieses PRINT

nicht durch das Fragezeichen abgekürzt werden kann. Außerdem darf kein Leerzeichen zwischen dem PRINT und dem #-Zeichen stehen.

PRINT#5,X schreibt zum Beispiel den Wert X in eine entsprechende, vorher geöffnete Datei (siehe ->sequentielle Datei).

Programm-Modus

Eigentlicher Sinn der BASIC-Befehle ist es, sie nicht ein einziges Mal im ->Direktmodus, sondern im Rahmen eines Programms zu verwenden. Dies nennt man dann Programm-Modus. Entscheidend ist dabei, daß zu Beginn jeder Befehlszeile eine Zahl steht. Die Programmzeilen werden von BASIC mit Hilfe dieser Zeilennummern geordnet. Die Betätigung der Return-Taste führt noch nicht zur Ausführung des Befehls, sondern nur zur Ablagerung im Programmspeicher. Mit LIST kann man sich einzelne oder alle Befehlszeilen zeigen lassen. Erst der Befehl RUN führt zur Abarbeitung in der Reihenfolge der Zeilennummern. Vorteil des Programm-Modus im Gegensatz zum Direktmodus ist die fast unbeschränkte Verknüpfbarkeit der Befehle zu langen Programmen, außerdem die Möglichkeit der Speicherung.

Programmzeilen kopieren

Leicht verdoppelt bzw. beliebig kopiert werden können Programmzeilen, indem man über ihre Zeilennummer die gewünschte neue Nummer schreibt und die Return-Taste betätigt. Achtung, dabei werden bestehende Zeilen überschrieben. Haben Sie also ein Programm

```
10 REM Erste Zeile
20 REM Zweite Zeile
```

und überschreiben in der ersten Zeile die 10 durch eine 20, so wird die vorher existierende Zeile 20 durch die verdoppelte erste Zeile überschrieben.

Programmzeilen löschen

Programmzeilen kann man einfach dadurch löschen, daß man ihre Nummer auf den Bildschirm schreibt und anschließend direkt die Return-Taste betätigt.

Quadratwurzel

Wenn Sie aus einer Zahl die Quadratwurzel ziehen wollen, so geht das mit Hilfe der SQR-Funktion in BASIC.

```
PRINT SQR(16) ergibt 4
PRINT SQR(17) ergibt als Wert 4.12310563
```

Da wird das Rechnen im Kopf schon etwas schwer!

READ

Mit dem Befehl READ können Sie Daten aus speziellen Programmzeilen, den sogenannten DATA-Zeilen, einlesen und einer Variablen zuweisen.

```
10 READ BETRAG
20 PRINT BETRAG
30 DATA 100
```

Der Vorteil liegt unter anderem darin, daß die DATA-Zeilen nicht direkt hinter dem READ-Befehl stehen müssen. Man packt sie meistens komplett, d.h. für alle Variablen, an den Schluß des Programms. Wichtig ist nur, daß die READ-Befehle in der richtigen Reihenfolge auf die DATA-Angaben zugreifen. Diese werden nämlich einfach aneinandergereiht, lediglich durch Komma getrennt:

```
100 DATA 100,220,34.5
```

Daran wird deutlich, daß man bei Werten in DATA-Zeilen besonders darauf achten muß, Punkt und Komma nicht zu verwechseln. Denn nach jedem Komma beginnt für BASIC ein neuer Wert. Und hier wäre es verhängnisvoll, wenn man aus Versehen die halbe Mark als ,5 eingetragen hätte statt als .5!

Übrigens: Die DATA-Zeilen dürfen wirklich durch nichts unterbrochen werden, auch nicht durch REM-Bemerkungen!

Versucht ein Programm mit READ mehr Werte zu lesen, als DATA-Werte vorhanden sind, so gibt es die Fehlermeldung "Out of DATA ERROR IN ..."

Rechnen mit BASIC

Siehe ->Grundrechenarten.

REM

Mit dem Befehl REM (für englisch: REMARK = Bemerkung) können Sie BASIC mitteilen, daß das, was dahinter steht, nur für Sie eine Hilfe sein soll und keinen Befehlscharakter hat. Sinnvoll sind solche Bemerkungen vor allem, um Hinweise zu Programmteilen zu geben, etwa: REM HAUPTPROGRAMM.

RETURN

Dieser Befehl markiert das Ende eines mit GOSUB angesprungenen Unterprogramms, siehe daher auch GOSUB. Trifft BASIC auf ein RETURN, ohne vorher den Befehl GOSUB erhalten zu haben, so meldet es "RETURN WITHOUT GOSUB ERROR IN ..."

Return-Taste

Die Return-Taste befindet sich rechts auf der Tastatur und ist etwas größer als die anderen Tasten. Im ->Direktmodus dient sie zum Abschluß einer Befehlseingabe, im ->Programm-Modus schließt sie eine Zeile ab und macht die Eingabe oder Änderung dieser Zeile gültig.

RND-Funktion

RND (X) ist eine Funktion in BASIC, mit deren Hilfe Sie Zufallszahlen produzieren lassen können. Dabei liegen diese zwischen 0 und 1. Es reicht aus, für X immer die Zahl 1 zu nehmen. Dem Zufall tut das keinen Abbruch.

```
FOR I = 1 TO 3 : PRINT RND(1) : NEXT I
```

bringt Ihnen z.B. die folgenden Zufallszahlen auf den Bildschirm.

```
.417980108
.829277551
.809331095
```

Wir haben diese Zahlen wirklich auf dem Bildschirm erhalten und brav abgeschrieben, obwohl dies einer der seltenen Fälle ist, wo niemand uns eine kleine Mogelei hätte nachweisen können. Das ist eben ein Vorteil des Zufalls.

Will man mit Hilfe der RND-Funktion etwa ganzzahlige Werte haben, muß man sie entsprechend (etwa mit 10) multiplizieren und ganzahlig machen (siehe ->Integervariable und/oder den Funktionsbefehl ->INT.)

RUN

Mit RUN (Return) wird ein Programm, das man erstellt oder geladen hat, gestartet.

Speichern eines Programms

Siehe ->SAVE.

SAVE

Mit Hilfe der Befehlszeile

```
SAVE "PROGRAMMNAME",8 (Return)
```

kann man ein Programm abspeichern. Aufpassen muß man dabei, daß nicht schon ein Programm mit diesem Namen auf der Diskette existiert. Dann meldet sich der C64 zwar auch mit einem READY, die Floppy aber ist mit diesen Verhältnissen äußerst unzufrieden und beginnt aufgeregt zu blinken. In diesem Fall sollte man das Programm entweder unter einem anderen, noch nicht benutzten Namen abspeichern oder aber den sogenannten Klammeraffen verwenden. Sie finden ihn zwischen der Taste P und der Taste mit dem Sternchen. Wir verwenden für das Symbol hier den Begriff. Wollen Sie also ein Programm unter demselben Namen, etwa in einer verbesserten Version abspeichern, so geht das mit dem folgenden Direktbefehl:

```
SAVE "KLAMMERAFFE:PROGRAMMNAME",8 (Return)
```

Zu beachten ist, daß der "Klammeraffe" nicht immer fehlerlos arbeitet. Ein sicherer Weg ist, das entsprechende Programm vor-

her zu löschen und neu abzuspeichern (siehe ->Löschen eines Programms auf Diskette)

Sequentielle Datei(en)

Bei sequentiellen Dateien werden (im Gegensatz zu READ und DATA) die Daten separat auf Diskette gespeichert und vom Programm eingelesen. Der Begriff bedeutet, daß die Daten in einer bestimmten Reihenfolge (sequentiell = aufeinander folgend) abgespeichert sind, die zu beachten ist.

Zwei zentrale Vorgänge sind bei der Arbeit mit sequentiellen Dateien zu unterscheiden: das Schreiben von Daten auf Diskette und das Einlesen. Für beide Zwecke muß die Datei auf eine entsprechende Weise "geöffnet" werden. Dazu wird der von verschiedenen Vorgängen her bereits bekannte OPEN-Befehl verwendet, wobei hier zwei zusätzliche Hinweise zu beachten sind.

Um eine Datei mit dem Namen MEINEDATEN anzulegen, kann man die folgende Programmzeile verwenden:

```
10 OPEN 5,8,5,"MEINEDATEN,S,W"
```

Wichtig ist die Zahlengruppe nach dem OPEN, die wir in dieser Form immer verwenden, und außerdem die durch Komma abgetrennten Buchstaben S und W. Das S steht dabei für "sequentiell", und das W steht für WRITE (= Schreiben). Damit weiß der C64, daß er Daten in Richtung Floppy schicken soll.

Der konkrete Schreibbefehl lautet dann z.B.:

```
50 PRINT#5,X
```

Dabei ist es wichtig, daß das Leiterzeichen direkt hinter dem PRINT steht und die vom CLOSE-Befehl her bekannte 5 wieder verwendet wird. Das X steht hier für eine numerische Variable. Genauso könnte mit X$ eine Stringvariable auf die Diskette geschrieben werden.

Nach dem Schreiben der Daten muß der Kanal wieder mit z.B.

```
70 CLOSE 5
```

geschlossen werden.

Wichtig beim "Öffnen" einer Datei zum Schreiben (strenggenommen handelt es sich dabei immer um das Anlegen einer Datei) ist, daß nicht schon eine entsprechende Datei auf der Diskette existiert. Sicherheitshalber kann man einen entsprechenden Löschbefehl vorschalten:

```
10 OPEN 1,8,15,"S:MEINEDATEN":CLOSE 1
```

Der umgekehrte Vorgang, das Einlesen von Daten von Diskette, funktioniert im Prinzip ähnlich, nur mit entsprechend abgeänderten Befehlen. Auch hier wird zunächst wieder ein Kanal zur Floppy geöffnet, mit dem Hinweis R für READ (= Lesen).

```
100 OPEN 5,8,5,"S:MEINEDATEN,S,R"
```

Anschließend werden die Daten geholt, z.B.:

```
120 INPUT#5,X
```

Hier sind dieselben Punkte zu beachten wie beim Schreiben der Daten (s.o.).

Ganz am Schluß muß der Kanal wieder wie oben geschlossen werden, etwa mit:

```
150 CLOSE 5
```

Wichtig ist, daß man nie versucht, mehr Daten von Diskette zu lesen, als man dort abgelegt hat. Zu diesem Zweck empfiehlt es sich, als erste Zahl immer die Anzahl der Daten abzuspeichern und diese Zahl entsprechend im Programm zu berücksichtigen.

Starten eines Programms
Siehe ->RUN.

Schleifen
Siehe ->FOR NEXT.

Sprites

Sprites stellen neben ASCII-Zeichen-Grafik und der hochauflösenden Grafik eine weitere und besondere Darstellungsmöglichkeit des C64 dar, die ihn berühmt gemacht und zu seinem großen Erfolg beigetragen hat.

Sprites können 24 Punkte breit und 21 Punkte hoch sein. Da in der Breite immer 8 Punkte in einen Speicherplatz passen, werden 63 Speicherplätze benötigt. Sie befinden sich in einem speziellen Speicherbereich. Dadurch können Sprites im Textmodus nicht nur auf dem Bildschirm stehenbleiben, sondern sogar durch einfaches Überschreiben nicht gelöscht werden. Sie scheinen vor dem normalen Bildschirm zu liegen.

"Angestellt" wird der Sprite-Modus durch die folgenden Programmzeilen:

```
30 V = 53248
40 POKE V,50
50 POKE V+1,50
60 POKE V+21,1
```

Ausgestellt wird er durch:

```
80 POKE V+21,0
```

Im Kapitel 9 finden Sie einen sogenannten Sprite-Generator, d.h. einen Programmteil, mit dem Sie leicht ein Sprite erstellen und es dann in Ihr Programm einbauen können.

Bis zu acht Sprites können verwaltet werden. Dabei muß man nur darauf achten, daß ihr Aussehen an der richtigen Stelle abgelegt und dies dem VIC ab Adresse 53248 mitgeteilt wird. Wenn Sprites dasselbe Aussehen haben, kann auf dieselben Speicherstellen zurückgegriffen werden.

Man kann Sprites mit einer gewünschten Farbe versehen. Dies geschieht, indem man in V+39 eine Zahl zwischen 0 und 15 schreibt, also beispielsweise

```
POKE V+39,1
```

für ein weißes, erstes Sprite. In die folgenden 7 Speicherstellen V+40, V+41 usw. kommen dann die Farben eventueller weiterer Sprites.

Eine besondere Möglichkeit stellt das Zusammentreffen von Sprites, ihre Kollision dar, für viele Spiele eine unverzichtbare Grundlage. Dabei wird in Speicherstelle V+30 ein Signal vom VIC gesetzt, dort erscheint dann eine Zahl ungleich 0, die Sie mit PEEK(V+30) erfragen können. Ebenso kann in V+31 ein Zusammenstoß eines Sprite mit einem sichtbaren Teil des Bildschirms geprüft werden.

STEP

Mit STEP können Sie bei einem ->FOR-NEXT-Befehl eine Schrittweite angeben: FOR LITER = 1 TO 30 STEP .2 bedeutet, daß immer um 0.2 erhöht wird. STEP kann auch negativ sein und bedeutet dann, daß von einem höheren Wert rückwärts bis zu einem anderen gezählt wird. Dann muß natürlich der zweite Wert kleiner sein.

STOP

Im Gegensatz zu END wird mit dem Befehl STOP in einer Programmzeile die Abarbeitung des Programms an dieser Stelle nicht beendet, sondern nur unterbrochen. Es wird dementsprechend auch kein READY, sondern BREAK IN ... gemeldet. Anschließend kann man mit CONT (Return) das Programm fortsetzen.

STRING TOO LONG

Taucht diese Fehlermeldung auf, haben Sie versucht, BASIC einen String "unterzujubeln", der länger als 255 Zeichen ist. Das wird nicht akzeptiert. Beachten Sie bitte, daß auch Leerzeichen in einem String als Zeichen mitgezählt werden.

Stringvariablen

Unter Stringvariablen versteht man solche ->Variablen, die für Namen und Begriffe stehen, wobei in Grenzen Zahlen integriert sein können, z.B. "Hauptstraße 41".

Kennzeichnend für Stringvariablen ist, daß sie am Ende ihres Namens direkt das Dollarzeichen angefügt bekommen und der eigentliche Name bzw. String natürlich in Anführungszeichen gesetzt wird:

```
PN$ = "HAUSHALT" (PN für Programmname)
```

Der Befehl (direkt oder im Programm) PRINT PN$ bringt dann den gewünschten Begriff auf den Bildschirm.

Eine Stringvariable kann leer ("") sein oder bis zu 255 beliebige Zeichen enthalten. Das können auch ganze Sätze sein:

```
S$ = "Ich habe heute keine Lust zu arbeiten!"
```

Dabei zählen die Leerzeichen bei der 255er-Grenze mit.

Die Inhalte von Stringvariablen können durch ein Pluszeichen miteinander verknüpft und erneut zugewiesen werden.

```
W1$ = "Ich sehe "
W2$ = "Detlef"
W3$ = W1$ + W2$
```

ergibt: "Ich sehe Detlef".

STR$

Mit diesem Befehl können Sie eine Zahl in einen String umwandeln. So ergibt STR$(123) den String "123". STR$ kann natürlich nicht nur auf Zahlen, sondern auch auf numerische ->Variablen angewandt werden. Das Gegenstück zu STR$ lautet ->VAL.

?SYNTAX-ERROR

Allgemeiner Hinweis von BASIC darauf, daß es mit einem Befehl oder einer Zeile nichts anfangen kann.

Teilen

Siehe ->Division.

Textmodus

Darunter versteht man den Zustand des C64, in dem er sich nach dem Einschalten befindet und bei dem er die normalen Tastaturzeichen produziert. Der Textmodus entspricht der von uns so genannten ->ASCII-Grafik.

TYPE MISMATCH

Diese Fehlermeldung wird ausgegeben, wenn Sie einem Variablentyp eine falsche Variable zuweisen. N$ = 40 z.B. führt zu dieser Fehlermeldung, denn BASIC erwartet einen String und keine Zahl.

?UNDEF'D STATEMENT ERROR IN ...

Diese Fehlermeldung tritt auf, wenn Sie sich etwa bei GOTO oder GOSUB auf eine Zeilennummer beziehen, die gar nicht exisitiert. Schauen Sie sich einfach einmal das Zielgebiet Ihres Sprungwunsches an! Vielleicht sind Sie selbst sehr erstaunt.

Unterprogramm

Unterprogramme dienen dazu, bestimmte Programmteile (Unterprogramme) von verschiedenen Stellen aus aufrufen zu können. Sie enden mit RETURN, haben aber keinen definierten Anfang. Dieser wird nur durch die Zeilennummer beim Aufruf mit "GOSUB Zeilennummer" (z.B. GOSUB 100) definiert. Beginnen Sie daher am besten Unterprogramme mit einer REM-Zeile. Ein Beispiel könnte so aussehen:

```
10 Z = 1
20 GOSUB 100 : REM Zahl ausgeben
30 Z = 2
40 GOSUB 100 : REM Zahl ausgeben
50 END
100 REM Unterprogramm zur Ausgabe der Zahl "Zahl"
110 PRINT Z
120 RETURN
```

Wichtig ist das END in Zeile 50 (sonst gibt es ein "RETURN without GOSUB"), und hilfreich sind die REM-Zeilen bei längeren Programmen.

VAL

Mit VAL können Sie einen String oder den Inhalt einer Stringvariablen soweit möglich in eine Zahl verwandeln, solange vorne Ziffern stehen. Beispiele dafür sind:

```
VAL ("123") ergibt 123
VAL ("123abc" ergibt 123
VAL ("Autor") ergibt 0 (kein Kommentar)
```

Das Gegenstück von VAL lautet STR$.

Variablen

Mit Variablen (wörtlich: Veränderliche) sind zusammenhängende Zeichengruppen gemeint, die als Platzhalter für bestimmte Größen in einem Programm verstanden werden können. In dem Beispiel VB = LI / KI * 100, das wir hier als Programmzeile verstehen wollen, stehen die drei Variablennamen VB, LI und KI für bestimmte Werte, die entweder eingegeben oder berechnet werden. Diese Werte für den Benzinverbrauch (VB), die getankten Liter (LI) und die gefahrenen Kilometer (KI) sind grundsätzlich "variabel", d.h. beliebig im Rahmen einer konkreten Berechnung veränderlich. Verbunden werden sie durch eine Formel. Aus dem Beispiel ergibt sich bereits, daß solche Variablen die wichtigsten Grundlagen eines jeden Programms sind. Man könnte sie auch als Schublade verstehen, in der jeweils ein bestimmter Wert, der sich wieder per Berechnung aus anderen Variablen ergeben kann, abgelegt wird.

Das Beispiel KO = KO + 20 (dabei steht KO für KOSTEN) zeigt, daß das Gleichheitszeichen in einer Formel mit Variablen nicht Identität ausdrückt, sondern eine Zuweisung. In diesem Fall wird die Schublade mit dem Namen "KO" mit dem Wert gefüllt, der vorher darin war, vermehrt um den Wert 20.

Man unterscheidet ->numerische Variablen und ->alphanumerische Variablen oder ->Stringvariablen. Erstere stehen für Zahlen und Zahlenverhältnisse, letztere für Namen und Begriffe, wobei in Grenzen Zahlen integriert sein können.

Für die Wahl von Variablennamen gilt folgendes:

- Sie dürfen zwar maximal zwei Bildschirmzeilen lang sein, vom C64 unterschieden werden aber nur die ersten beiden Zeichen. D.h. KILOGRAMM und KILOMETER sind für ihn identisch. Daher empfiehlt es sich, aussagekräftige und nicht wiederholte Kurz-Variablennamen zu verwenden.
- Sie sollten nur Buchstaben (a - z) und Ziffern (0 - 9) enthalten.
- Sie dürfen nicht mit einer Zahl beginnen.
- Sie dürfen kein BASIC-Befehl sein und auch bestimmte Zeichengruppen nicht enthalten. Dabei sind ST, ON, TI von besonderer Bedeutung, weil sie sich leicht in STARTZAHL, TELEFON oder TITELANZAHL verstecken.

Bei Stringvariablen gilt zusätzlich, daß sie am Ende ihres Namens direkt das Dollarzeichen angefügt bekommen und der eigentliche Name bzw. String natürlich in Anführungszeichen gesetzt wird:

```
PN$ = "HAUSHALT" (PN steht hier für Programmname)
```

Der Befehl (direkt oder im Programm) PRINT PN$ bringt dann den gewünschten Begriff auf den Bildschirm.

Hingewiesen sei hier noch auf die ->Integervariablen, die alle Werte als ganzzahlig verstehen.

Für alle numerischen Variablen gilt, daß sie zunächst solange den Wert 0 enthalten, bis ihnen ein anderer Wert zugewiesen wird. Bei Stringvariablen ist dementsprechend am Anfang der Leerstring ("") vorgegeben.

Verschachtelte Schleifen

Man spricht von verschachtelten Schleifen, wenn man in einer FOR-NEXT-Schleife noch eine andere FOR-NEXT-Schleife laufen läßt, wobei beide natürlich unterschiedliche Lauf- oder Zählvariablen haben müssen. Ein entsprechendes Programm könnte zum Beispiel wie folgt aussehen:

```
10 PRINT "CLR"
20 FOR I = 1 TO 3
30 PRINT "*"
40 FOR J = 1 TO 3
50 PRINT "="
60 NEXT J
70 NEXT I
```

Es ergibt sich ein senkrechtes Muster, bei dem immer auf ein Sternchen drei Gleichheitszeichen folgen, sicherlich von beschränktem künstlerischen, dafür aber von um so höherem Erklärungswert.

In diesem Falle beginnt in Zeile 20 die äußere Schleife mit der Laufvariablen I. Sie wird natürlich auch erst am Schluß, in Zeile 70, geschlossen. In Zeile 40 beginnt die innere Schleife mit der Laufvariablen J. Sie wird als erste, nämlich in Zeile 60, auch wieder geschlossen. Dieses Einhalten der Reihenfolge ist bei verschachtelten Schleifen sehr wichtig.

Verzögerungsschleife

Mit einer Verzögerungsschleife kann eine zu schnelle Abarbeitung eines Programms (beispielsweise bei der Bildschirmausgabe) künstlich verlangsamt werden. Ein Beispiel dafür wäre:

```
10 FOR I = 1 TO 1000: NEXT I
```

Man nennt solche Schleifen auch leere Schleifen, weil innerhalb der Schleife keine Anweisungen erfolgen.

VIC

Abkürzung für Video-Controller. Siehe dazu die Kapitel: Hochauflösende Grafik und Sprites.

Warteschleife

Siehe ->Verzögerungsschleife.

Wurzel ziehen

Siehe ->Quadratwurzel.

Stichwortverzeichnis

\# 142

Abbruch 209
Alphanumerische Variablen 58
Arbeitsspeicher 23
Array 138, 209
ASC 95
ASCII 93
ASCII-Grafik 93, 114, 210
ASCII-Werte 94
ASCII-Zeichen 210
Ausblicke 193
Ausdrucken 45
Ausdrucken von Programmen 46
Änderung von Befehlseingaben 27

Balken 107
Balkengrafik 109, 120
BALKENGRAFIK (Programm) 134, 148
Balkenwerte 117
Ballon 179
Ballonspiel 184
BASIC 23, 32, 195, 210
BASIC-Befehle 69
BASIC-Erweiterungen 196
BECKERbasic 64 196
Benzinkostentabelle 77
Bildschirm löschen 95, 211
Bildschirmfarben (einstellen) 211
Binär 211
Bit 212
Byte 212

CHR$ 95, 212
CLR 212
CMD1 46
Compiler 197

Computersprachen 212
CONT 212
CONTROL-Taste 212
Crashtest 183
Cursor 24, 213
Cursor Home 95
Cursor-Bewegung 213

DATA 127, 213
Datasette 20
Datei 213
Dateinamen 213
Daten 127
Datenspeicherung 143
Del-Taste 214
DIM 132, 214
Direktmodus 37, 216
Disketten 20
Diskettenlaufwerk 20
Division 215
Dreieck 96
Drucken 72, 215

Einfügen 216
Einfügen von Befehlszeilen 27
END 99, 217
Endlosschleife 169
Exponentialschreibweise 217

Farben 154, 182
Fehlerquellen 199
Felder 131, 217
File 218
Fließkommazahl 85
Floppy 20
Floppybuch 194
FOR NEXT 74, 218
Formatierbefehl 33
Formatieren 218

Formeln 164
Fragezeichen 218

Gehaltserhöhung 66
GEOS 16, 196
Geschwindigkeit 197
GET 95, 219
GOSUB 99, 219
GOTO 69, 220
Grafik 93
Grafikmodus 156
Grafikzeichen 24
Großbuchstaben 18
Grundrechenarten 220

Hauptprogramm 220
HIRES-Grafik 153
Hochauflösende Grafik 153
Hochpfeil 220

ID-Nummer 36
Identifikationsnummer 36
IF THEN 70, 220
Inhaltsverzeichnis 221
Inhaltsverzeichnis der Diskette 34
Initialisierung 223
Initialisierungsteil 222
INPUT 42, 218, 222
INPUT# 222
Inst-Taste 222
INT 82, 223
Integervariablen 85, 223

Kassettenrecorder 20
Kleinbuchstaben 18

Laden 37, 224
Lexikon 209
LIST 223
Literatur 193

LOAD 38, 224
Löschen 37, 224

Malen 101
Malprogramm 162
Maschinensprache 195
Mathematik 164
Mathematische Funktionen 164
Mehrwertsteuer 63
Modus 19
Modus I 153
Modus II 154
Monatsbalken 119

NEW 225
Numerische Variablen 58, 225

PEEK 225
POKE 172, 226
Potenzieren 226
PRG 142
PRINT 26, 59, 226
PRINT# 227
Programm 23, 32
Programm-Modus 37, 228
Programmpuzzle 163
Programmzeilen 228
Prozentzeichen 85
Punkte setzen 159

Quadratwurzel 229

Rabatt 64
READ 127, 229
Rechnen 28, 230
REM 80, 230
RETURN 99, 219, 230
Return-Taste 230
RND 230
RUN 40, 231

SAVE 38
Schleifen 233
Scratch 41
Semikolon 59
SEQ 142
Sequentielle Datei 139, 232
Sicherung 63
Speichermöglichkeit 20
Speichern 37, 231
Spritemaske 179
Sprites 171, 234
Sprites bewegen 181
Starten 233
STEP 81, 235
STOP 235
STR 236
Stringvariablen 56, 235
Suchen 37

Tabelle 71
Tastatur 17
Teilen 236
Textmodus 153, 237

Umschaltung der Tastatur 18
Unterprogramme 99, 237
UPZEICHEN (Programm) 101

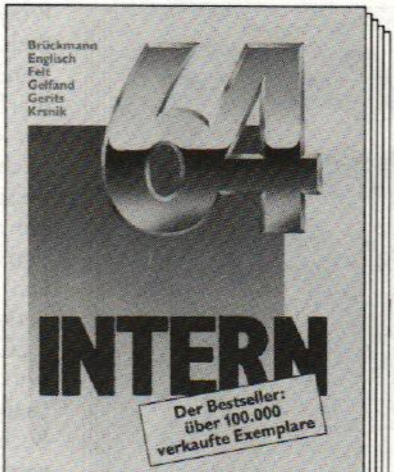
VAL 238
Variablen 53, 79, 238
Variablennamen 54
Vergleich von Daten 73
Verschachtelte Schleifen 110, 239
Verzögerungsschleifen 169, 240
VIC 172, 240
Video-Controller 167, 172

Warteschleife 240
Wurzel 240

Zahlenraten 83
Zeichenkette 59
ZUFALLSZAHL (Programm) 90
Zufallszahlen 84

C64 Intern – der Bestseller unter den C64-Büchern.

C64 Intern – für dieses Buch ist keine lange Vorrede nötig. Der Bestseller mit über 100.000 verkauften Exemplaren gehört einfach neben jeden C64.

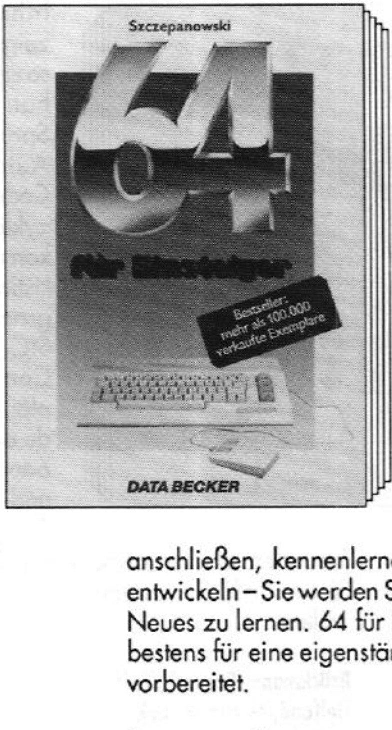


Ein kleiner Blick ins Inhaltsverzeichnis zeigt, was dieses Buch so erfolgreich gemacht hat: Soft-Scrolling, Sprungvektoren und Autostart, Illegal-Codes und deren Taktzyklen, zeilenweise kommentiertes ROM-Listing, Interrupt-Programmierung, BASIC-Intern, original Commodore-Schaltpläne, die Unterschiede der verschiedenen 64er-Modelle und und und. Verabschieden Sie sich von Ihren Freunden: Über 600 spannende Seiten warten auf Sie. Mit allem, was engagierte 64er-Fans wissen wollen.

Brückmann/Englisch/Felt
Gelfand/Gerits/Krsnik
64 Intern
648 Seiten, inklusive Diskette, DM 69,-
ISBN 3-89011-000-2

Mit Schwung und Laune Neues lernen.

Anfangen und gleich loslegen – das wünscht sich jeder, der in die Computerei einsteigt. Mit „64 für Einsteiger" geht das ganz problemlos. Vom Anschluß bis zum ersten Programm. Systematisch und leichtverständlich lernen Sie hier Ihren neuen Rechner kennen. Mit zahlreichen Anwendungsbeispielen, Erklärungen zur hochauflösenden Grafik und natürlich mit einer detaillierten Einführung in GEOS 2.0 deutsch. Ein umfangreiches Lexikon macht das Buch auch dann noch wertvoll, wenn Sie längst kein Einsteiger mehr sind. Aufstellen, anschließen, kennenlernen, das erste Programm entwickeln – Sie werden Spaß daran finden, etwas Neues zu lernen. 64 für Einsteiger – und Sie sind bestens für eine eigenständige Arbeit am Rechner vorbereitet.



Szczepanowski
64 für Einsteiger
251 Seiten, DM 29,-
ISBN 3-89011-010-X

Alles für Ihre ersten Schritte mit GEOS.

Ein Einstieg, wie er im Buche steht: Von der Arbeitsdiskette bis zum ersten Brief lernen Sie Schritt für Schritt all die Möglichkeiten von GEOS kennen – anhand zahlreicher ausführlicher Beispiele. Wieder und wieder werden Sie dabei auf neue, überraschende Ideen und Anregungen stoßen. Und eh' Sie sich versehen, arbeiten Sie problemlos mit dem Desktop, setzen die verschiedensten Anwendungen ein und beherrschen auch die Zusatzprogramme wie GeoPaint und GeoWrite. Bei alledem werden die GEOS-Versionen 1.2 und 1.3 deutsch besonders ausführlich behandelt. GEOS für Einsteiger – geeignet sowohl für den C64 als auch für den C128.

Tornsdorf
GEOS für Einsteiger
252 Seiten, DM 29,-
ISBN 3-89011-275-7

Das Nachschlagewerk für zukünftige GEOS-Profis.

Das große GEOS-Buch – unentbehrliche Grundlage für alle zukünftigen GEOS-Profis. Hier finden Sie alles, was es zu GEOS zu sagen gibt: Wie ist das GEOS-File-Format aufgebaut? Wie erstelle ich eine bootfähige Sicherheitskopie? Wie schreibt man Programme mit GEOS-Eigenschaften? Wie bekomme ich einen Maschinensprache-Monitor in GEOS? Sonst noch Fragen? Einfach nachschlagen. Das große GEOS-Buch wird Ihnen alle Fragen zu GEOS Schritt für Schritt beantworten. Natürlich mit jeder Menge Tips & Tricks für Ihre tägliche Arbeit. Bei alledem werden selbstverständlich die GEOS-Version 1.2 und 1.3 besonders intensiv behandelt. Das große GEOS-Buch – geeignet für den C64 und C128.

Kerkloh/Tornsdorf
Das große GEOS-Buch
424 Seiten, DM 49,-
ISBN 3-89011-208-0

Alle Tricks für einen optimalen Softwareschutz.

Schützen Sie Ihre Programme mit einem wirkungsvollen, optimalen Software-Schutz. Egal ob BASIC- oder Assembler-Programme, ob Kassetten oder Disketten-Programme – das Know-how dazu haben Profis im Anti-Cracker-Buch zusammengefaßt. Hier wird alles systematisch und ausführlich dargestellt – Sie brauchen also keinerlei Vorkenntnisse, wenn die Rede ist von Illegal Opcodes, Nutzung der Tracks 36 bis 41, Half-Tracks, Killertracks, Einzelschritt-Decodierer und und und. Sie erfahren sehr schnell, was sich im Einzelnen hinter diesen Begriffen verbirgt und was sie in der Praxis bedeuten. Daneben lüftet dieses Buch endlich die Geheimnisse wirkungsvoller Kopierschutzverfahren und zeigt Softwarehäusern, wie die erfolgreichen Cracker vorgehen.

Gelfand/Felt/Strauch/Krsnik
Das Anti-Cracker-Buch
379 Seiten, DM 39,-
ISBN 3-89011-253-6

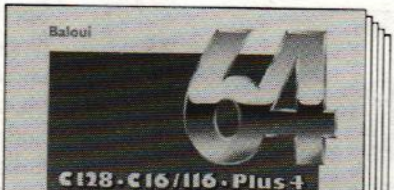
Der perfekte Einsatz Ihrer Floppy-Station.

Mt der Floppy läßt sich weitaus mehr machen als nur laden und starten. Man muß sich lediglich ein wenig auskennen. Was Sie alles aus Ihrer Floppystation herausholen können, zeigt Ihnen das Commodore Floppybuch. Vom Einstieg bis zur Programmierung der Floppy in BASIC. Hier finden Sie alles über den Aufbau von Disketten, zu den einzelnen Dateitypen, zu den Systembefehlen und natürlich auch zu den verschiedenen Fehlermeldungen der unterschiedlichen Floppystationen. Was Sie über Ihre Floppy wissen sollten, finden Sie in diesem Buch – ob Einsteiger oder Profi und egal mit welcher Floppystation Sie arbeiten: der 1541, der II/41 oder der C/70/71/81.

Schönleber
Das Commodore-Floppybuch
240 Seiten, DM 29,-
ISBN 3-89011-269-2

Maschinensprache verständlich für jedermann.

Endlich einmal kein unverständliches Lehrbuch, sondern ein Buch, mit dem wirklich jeder, der sich dafür interessiert, schnell und einfach Maschinensprache lernen kann.



Ohne das übliche Fachchinesisch wissen Sie schon bald, was ein professionelles Programm ausmacht: BASIC-Routinen heranziehen, Befehle und Strukturen vergleichen und schließlich selbst in Assembler umsetzen – durch dieses Konzept sind Sie sehr schnell in der Lage, die Vorteile dieser Sprache für eigene Anwendungen voll zu nutzen. Eine echte Chance für alle Assembler-Interessierten, denen diese Sprache bisher zu schwierig erschien. „Sprechen" Sie die Sprache der Profis – mit Maschinensprache für Einsteiger. Geeignet für den C64, den C128 und auch für die ganz Kleinen (C16/C116/Plus4).

Baloui
Maschinensprache für Einsteiger
345 Seiten, DM 29,-
ISBN 3-89011-182-3

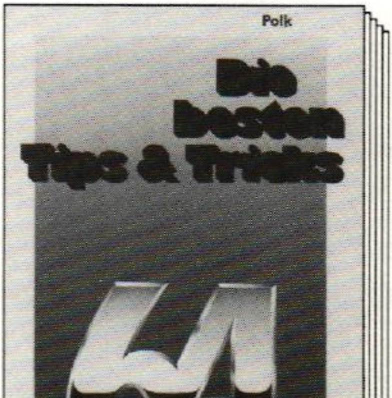
Das Supergrafik-buch zum Commodore 64.

Das Supergrafikbuch zum C64 – ein hundertprozentiges Buch. Mit der Befehlserweiterung Supergrafik auf Diskette, die über 100 weitere, leistungsstarke Befehle, 15 Sprites, Text und Grafik gleichzeitig, Hires- und Multicolor-Grafik bietet. Im Buch selber finden Sie dann natürlich zu jedem dieser Befehle Top-Beispielprogramme, die Ihnen zeigen, was Ihr C64 grafisch so alles „draufhat". Außerdem: Funktionsplotter, 3-D-Grafik, Hardwaregrundlagen und eine vollständige Dokumentation des Source-Codes! Das Supergrafikbuch zum Commodore 64 – ein Muß für jeden 64er-Fan, der auch grafisch alles aus seinem Rechner holen will.

Plenge
Das Supergrafikbuch zum Commodore 64
Hardcover, 726 Seiten
inklusive Diskette, DM 49,-
ISBN 3-89011-213-7

Die besten Tips und Tricks zum Commodore 64.

Jetzt noch mehr Tips & Tricks rund um Ihren C64. Die besten – als Einzeiler, Kurzprogramme oder Peeks und Pokes. Ob zur Datasette, zum Speicher oder zur Floppy. Ob zu BASIC oder zum Softwareschutz. Zu Grafik oder Sound. Oder auch zum Schönsten am C64: den Spielen. Zu allen Bereichen zeigt Ihnen dieses Buch, wie Sie Ihre Arbeit optimieren können. Mit den richtigen Kniffen zur rechten Zeit. Zusammengestellt von einem langjährigen 64er-Experten. Ersparen Sie sich langes Suchen in einschlägigen Zeitschriften, die besten Tips und Tricks für Ihre Arbeit stehen in diesem Buch. Holen Sie alles aus Ihrem C64. Mit den Tips & Tricks eines echten Profis.



Polk
Die besten Tips & Tricks
288 Seiten, DM 29,-
ISBN 3-89011-281-1

DAS STEHT DRIN:

Sie wollen die vielfältigen Möglichkeiten Ihres neuerworbenen C64 möglichst schnell kennenlernen. Der Schlüssel dazu heißt BASIC. Daß die Einarbeitung darin nicht schwierig sein muß, ja sogar Spaß machen kann, zeigt dieses Buch. Schritt für Schritt werden Sie in die Welt des C64 und seines BASIC eingeführt. Durch Zusammenfassungen werden die neuerworbenen Fähigkeiten zum sicheren Besitz. Das umfangreiche Lexikon am Schluß macht das Buch für Sie auch noch wertvoll, wenn Sie kein Einsteiger mehr sind.

Aus dem Inhalt:

- BASIC an einem Abend: Der schnelle Weg zum ersten Programm
- BASIC an einem Wochenende: Variablen (Bedeutung und Arten, Anwendungsbeispiele aus der alltäglichen Praxis)
- Grundbefehle für die Erstellung von Programmabläufen – GOTO, IF THEN, FOR NEXT
- Der Weg zum Profi-Programm – Einsatz von Unterprogrammen
- Zahlreiche Anwendungsbeispiele für Programme
- Grafik zum Nulltarif: einfache Grafik (von Phantasiebildern bis hin zu Balkendiagrammen)
- Datenspeicherung innerhalb von Programmen und auf Diskette – sequentielle Dateien
- Hochauflösende Grafik
- SPRITES und dazu ein Spielprogramm (Landung eines Ballons)
- Pannenhilfe – Zusammenstellung von Hilfen bei Problemen
- Lexikon – umfangreiche, leichtverständliche Begriffserklärungen zum Nachschlagen

UND GESCHRIEBEN HABEN DIESES BUCH:

Manfred Tornsdorf arbeitet seit Jahren mit dem C64 und kennt ihn in- und auswendig. Von ihm stammt unter anderem auch das „Große GEOS-Buch". Helmut Tornsdorf kennt besonders die Probleme von Einsteigern und hat die Erfahrungen zweier weiterer Einsteigerbücher in dieses Buch eingebracht.

ISB N 3-89011-246-3 DM +029.00

DM 29,–
ÖS 226,–
sFr 27,–

02900

DATA BECKER

9 783890 112466